DESIGN AND PERFORMANCE ANALYSIS OF COORDINATED CHECKPOINTING ALGORITHMS FOR DISTRIBUTED MOBILE SYSTEMS

Surender Kumar [1],R.K. Chauhan[2] and Parveen Kumar[3]

[1]Deptt. of I.T, Haryana College of Tech. & Mgmt. Kaithal-136027(HR), INDIA
Ssjangra20@rediffmail.com

[2]Deptt. of Computer Sc & Application, Kurukshetra University, Kurukshetra(HR),India
rkcdcsa@gmail.com

[3]Department of Computer Application, MIET, Meerut(U.P), India
Pk223475@gmail.com

ABSTRACT

Checkpointing is an efficient fault tolerance technique used in distributed systems. Mobile computing raises many new issues, such as high mobility, lack of stable storage on mobile hosts (MHs), low bandwidth of wireless channels, limited battery life and disconnections that make the traditional checkpointing protocols unsuitable for such systems. Several checkpointing algorithms have been reported in the literature. In this paper, we analyze some of existing coordinated checkpointing algorithms on the basic of blocking time, synchronization message overhead, number of processes required to checkpoint, number of useless checkpoint, piggybacked information messages onto computation messages and concurrent execution. We also proposed an efficient checkpointing algorithm to reduce the checkpointing overheads. Our checkpoint algorithm does not have any synchronization message overhead as it uses time to indirectly coordinate to create the consistent cut in distributed mobile system without increasing the number of checkpoints..

KEYWORDS

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

1. INTRODUCTION

Recent years have witnessed rapid development of mobile communications and become part of everyday life for most people. In the future, we will expect more and more people will use some portable units such as notebooks or personal data assistants. With increasing use small portable computers, wireless networks and satellites, a trend to support "Computing of the move" has emerged. This trend is known as mobile computing or "anytime" or "anywhere" computing. This enables the user to access and exchange information while they travel, roam in their home environments, or work at their desktop computers. Mobile environment contains both fixed and mobile hosts interconnected by a backbone network. Thus, recent advances in technology and mobile devices (e.g., laptop PCs with wireless connections, PDAs, etc.) have made the mobile computing affordable.
The mobile hosts have several characteristics that make them different from fixed host. So any checkpointing approach for fault tolerant in mobile environment should consider these distinguishing features in their application. Presence of following characteristics we distinguish between distributed system and mobile distributed system.

- Host Mobility
- Limited Battery power
- Frequently Disconnection

- Limited Bandwidth on wireless link
- High bandwidth variability
- Lack of Stable Storage on MH
- Different types of failure
- Limited geographical area
- Handoff

Numerous checkpointing and rollback recovery protocols have been proposed and studied extensively for distributed systems in the past years. However, little attention has been devoted to checkpointing mobile distributed systems. Now a day's wireless networks, and mobile devices become pervasive, it is necessary to extend the capability of checkpointing to wireless and mobile environment. Due to the above unique characteristics of mobile environments, it is not appropriate to directly apply checkpointing and recovery protocols designed for distributed systems to mobile distributed systems. Checkpointing and rollback protocol for mobile distributed systems must consider these above unique characteristics. Otherwise, the protocols may not perform correctly or efficiently.

2. SYSTEM MODEL

A distributed system is a collection of computers that are spatially separated and do not share a common memory. The processes executing on these computers communicate with one another by exchanging messages over communication channels. The messages are delivered after an arbitrary delay.

A mobile distributed system is a distributed system where some of the processes are running on mobile hosts (MHs)[5]. It consists of Static Hosts (SHs), Mobile Hosts (MHs) and the Mobile Support Stations (MSSs). So, the mobile distributed system can be considered as consisting of "n" MHs and "m" MSSs. The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency. Similarly, the wireless network within a cell ensures FIFO delivery of messages between an MSS and a local MH. The links are FIFO in nature. An MH communicates with other nodes of system via special nodes called mobile support station (MSS).An MH can directly communicate with an MSS only if the MH is physically located within the cell serviced by MSS. A static node that has no support to MH can be considered as an MSS with no MH. A cell is a geographical area around an MSS in which it can support an MH .An MH can change its geographical position freely from one cell to another cell or even area covered by no cell .At any given instant of time an MH may logically belong to only one cell; its current cell defines the MH's location and the MH is considered local to MSS providing wireless coverage in the cell. If an MH does not leave the cell, then every message sent to it from local MSS would receive in sequence in which they are sent.

Base Station (BS) provides the wireless environment within the cell. It acts as a mediator between MHs and Base station controller (BSC) i.e wired and wireless networks. It is connected to MH via wireless link and to BSC via a high speed wired link. Two or more MHs are controlled by BS, two or more BSs are controlled by a BSC and similarly Mobile Support Station (MSS) will control two or more BSC's. Due to mobility, the MH may cross the boundary between two cells; this process is known as handoff. The handover from one BS to another within the BSC region are handled by the BSC. MSS is high performance digital ISDN switch and is equipped with Home Location Register (HLR) and Visitor Location Register (VLR) for storing the location information of the mobile hosts. HLR is the master subscriber data base that contains details of each mobile host (MHs) like pertinent user information, including address, account status, and preferences etc. VLR act as a temporary subscriber data base database maintained by a MSS to track users who are roaming in that mobile service provider's area.

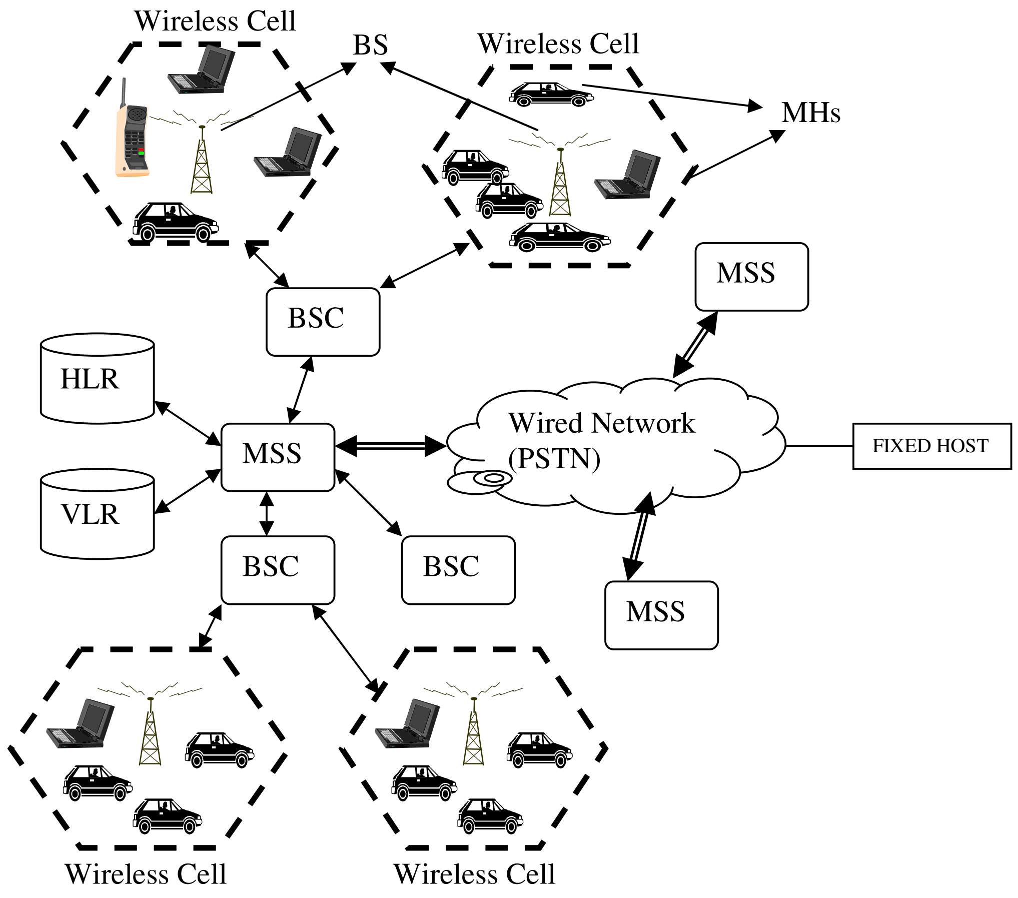

Figure 1. Reference Architecture Mobile Distributed Systems

Message communication from an MH says MH_1 to another MH says MH_2 occurs as follows .MH_1 first sends the message to its local MSS says MSS_1 using wireless link .MSS_1 forwards it to MSS_2, the local MSS of MH_2 via a fixed network. MSS_2 then transmit it to MH_2 over its wireless network. However location of MH_2 may not be known to MSS_1. So MSS_1 may require to first determining the location of MH_2 [23]. A node in the distributed system may fail during checkpointing. We assume all the failure to be fail-stop in nature. When a node fails, the contents of its primary memory are lost.

2.1 System Failure

The computing node in distributed system may fail. A system failure occurs when the processor fails to execute [34]. To handle the failure systems periodically saves the state in stable storage. At the time of failure system restart from its valid state. These failures can be classified in two different categories.

2.1.1 Hard failures

Hard failures consider as permanently failure or complete loss of connectivity from mobile node. This type of failure is non-voluntary in nature and processes stops any further actions forever such as falls, breaks, lost or stolen.

2.1.2 Soft Failure

Soft failures do not permanently damages the mobile host. In such case mobile host informs to MSS prior to its occurrence such as battery discharge, disconnections or operating crashes.
These two distinct types of failure can be handled by using checkpointing. The protocol use soft checkpoints which stored locally to tolerate soft failure and hard checkpoints are stored in the stable storage of MSS to tolerate the failures. 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 exchanges. Hard checkpoints have to be sent through the wireless link, and then through the backbone network, until they are stored in stable storage.

3. ROLLBACK RECOVERY MECHANISM

A rollback-recovery mechanism consists of three parts: checkpointing, fault detection and failure recovery. During checkpointing the state of a system periodically saved. When a failure occurs, processes have to rollback to their latest checkpointed state and continue execution from that state The main issues in rollback recovery are to minimizing the work to be undoing and to begin the rollback recovery as soon as possible for each process which must rollback and minimal required information of the state should be saved so that process can be restarted in case of an error.
Two main approaches of rollback recovery for the solutions to the problem of node failure are:

- Log based rollback recovery
- Checkpointing based rollback recovery

3.1 Log-Based rollback Recovery Mechanism

In log-based recovery, sending message history of processes since last checkpoints, are kept in main memory [7]. In case of a failure, a process can ask fault-free processes the needed messages. “Spooling” can be performing if volatile message logging takes too much memory space. In message logging protocols, each process periodically records its local state and logs the messages that it receives after having recorded that state on stable storage. When a process crashes, a new process is created in its place. The new process is given the appropriate recorded local state, and then the logged messages are replayed in the order the process originally received them. All message-logging protocols require that once a crashed process recovers, its recovered state is consistent with the states of the other processes [7]. Pessimistic logging, Optimistic and Casual Logging are three types of logging protocols [7].

3.2 Checkpoint-Based Rollback Recovery Mechanism

In checkpointing based rollback recovery is a well-established technique to deal with process failures and increase the system reliability and fault-tolerance in distributed systems [23]. In this approach, the state of each process in the system is periodically saved on stable storage, which is called a checkpoint of a process. To recover from a failure, the system restarts its execution from a previous error-free, consistent global state [3]. 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 processes exchange information with each other through messages. 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 [3]. There are several applications of checkpointing including: rollback recovery, playback debugging, process migration, job swapping and load balancing [22].

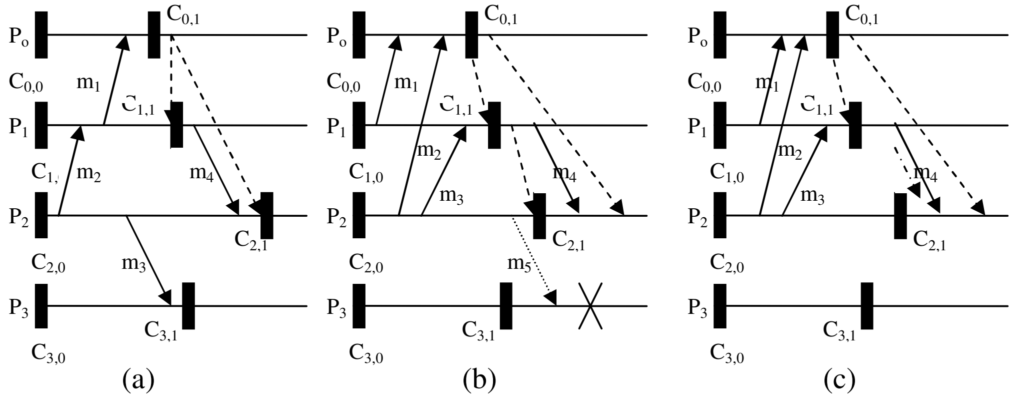

Figure 2. Non-blocking coordinated checkpointing: (a) with orphan, domino effect and checkpoint inconsistency; (b) with FIFO channels and lost message; (c) with non-FIFO channels.

3.2.1 Checkpointing Related Notations

Lamport's 'happen before' relation: (i) if a and b are two events occurring in the same process and if a occurs before b, then $a \rightarrow b$ (ii) if a is the event of sending a message and b is the event of receiving the same message in another process then, $a \rightarrow b$.

Orphan message and domino effect: Consider the system activity illustrated in figure 2(a). P_0, P_1, P_2 and P_3 are four processes that cooperate by exchanging information (shown by the arrows). Each symbol '' marks a recovery point to which a process can be rolled back in the event of a failure. If process P_2 is to be rolled back, it can be rolled back to the recovery point $C_{2,1}$ without effecting any other process. Suppose that P_1 fails after sending message m_4 and rolled back to $C_{1,1}$. In this case, the receipt of m_4 is recorded in $C_{2,1}$, but the sending of m is not recorded in $C_{1,1}$. Under such circumstances, m_4 is referred to as an *orphan message* (messages whose *receive* events are recorded in the states of the destination processes but the *send* events are lost) and process P_2 must also roll back because P_1 interacted with P_2 after establishing its recovery point $C_{1,1.}$ So this effect, where rolling back one process causes one or more other processes to roll back, is known as *domino effect* [34]. The domino effect is caused by *orphan message*, which themselves are due to rollbacks [33].

Lost message: Such a message, whose *send* event is recorded in the state of the sender process but the receive event is lost is called lost message as m_5 in figure 2(b) is a lost message.

Local checkpoints: In distributed systems, all processes may take a local checkpoint independently at any time during the execution. The process of saving local state is called local checkpointing [34]. The local checkpoints of different processes are not coordinate to form a global consistent checkpoint state. In figure 1(a) $C_{0,0}$, $C_{1,0}$, $C_{2,0}$, $C_{3,0}$, $C_{3,1}$ are the local checkpoints.

Global checkpoints: A set of local checkpoints, with one checkpoint for every process, is said to be *Consistent Global Checkpointing State* (CGS), if it does not contain any *orphan message* or *lost message*. However missing message are acceptable in GCS, if messages are logged by sender [36]. In figure 1(a) checkpoint state $C_{0,1}$, $C_{1,1}$, $C_{2,2}$ show an inconsistent checkpoints state due to orphan message m_4 but in figure 1(b) these processes show the consistent global state as there is not any *orphan* and *lost messages*.

FIFO Vs Non-FIFO channel: In FIFO system, checkpoint request play an important roll to determine the consistent global state. A FIFO system ensures that all messages sent after a checkpoint request on a channel will be delivered after the checkpoint request [37]. Hence, if channels are FIFO, process send first post-checkpoint message on each channel by checkpoint request before sending the message [7], as illustrated in figure 2(b). In non-FIFO system, the problem of global snapshot recording is complicated because a checkpoint request can not be used delineate messages into those not to be recorded in the global state [37]. If channels are non-FIFO, the checkpoint request can be piggybacked on every post-checkpoint message as in figure 2(c) [35].

Forced /Induced Checkpoint: A checkpoint that is forced due to receive of a message.

Useful and Useless Checkpoints: if a forced checkpoints is converted into tentative checkpoint after receiving the checkpoint request and become the member of global state is called useful checkpoint else it become useless checkpoint.

Directly and transitively dependent: A process P_i is in its y^{th} checkpoint interval *directly depends* on the process P_j on its x^{th} interval if P_j sends a message m after taking checkpoint $C_{j,x}$ and P_i receives it after taking the checkpoint $C_{i,y}$. Process P_i *transitively depends* on the process P_j if it depends *directly depends* on some processes P_m and P_m depends on P_j [19]. In figure 2(a) process P_0 is directly depends on process P_1 due to m_1 and transitively depends on P_2 as P_2 sends message m_2 to P_1 on which P_0 directly depends.

Minimum set: if P_i initiate its$(x+1)^{th}$ checkpoint then the set of processes on which P_i depends(directly or transitively) in its x^{th} checkpoint is minimum set[19]. In figure 2(a) on process P_0 has the processes in minset $\{P_0, P_1, P_2\}$ as P_0 directly depends and P_2 transitively depends upon P_0.

Fault tolerance: The ability of a system to perform with the presence fault.

Recovery line: A recovery line is a line which connects all the local checkpoints of a consistent global checkpointing state. If a failure occurs, then a system is requires to rollback to the latest available consistent global state.

3.2.2Checkpoint Algorithms Assumptions

Checkpoint algorithms assume the following characteristics for the distributed system [7]:

(i) Processes do not share memory and communicate by exchanging messages through communication channels.
(ii) Channels are FIFO in nature.
(iii) When a process fail, in such case it loses its volatile state and stops its volatile state and stops execution according to the fail-stop model.
(iv) Communication failures do not partition the network.
(v) Channels can loss messages. However, they are made virtually lossless and order of the messages is preserved by some end-to-end transmission protocol. Message sequence numbers may be used to preserve the order.
(vi) Processes are piecewise deterministic in the sense that from the same state, if given the same inputs, a process executes the same sequence of instructions.
(vii) Processes can save their state on stable storage to survives from failures during failure-free execution and can be used for recovery.

3.2.3 Types of Checkpointing

There are three flavors of checkpointing based recovery protocols:

1) Coordinated or Synchronous checkpointing
2) Uncoordinated or Asynchronous checkpointing
3) Communication induced or Quasi-Synchronous or Hybrid Checkpointing

We first explain the uncoordinated, communication induced checkpointing algorithms in short and at last coordinated checkpointing algorithms in details with comparative study.

1) Independent/Uncoordinated/Asynchronous Checkpointing

In independent checkpointing, processes do not synchronize their checkpointing activity and processes are allowed to records their local checkpoints in an independent way. After a failure, system will search a consistent global state 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. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [7]. Acharya-Badrinath[5] were the first who present a uncoordinated checkpointing algorithm for mobile computing systems. In their algorithm, an MH takes a local checkpoint whenever a message reception is preceded by a message sent at that MH. If the *send* and *receive* of messages are interleaved, the number of local checkpoints will be equal to half of the number of computation messages, which may degrade the system performance.

2) Quasi-synchronous/Hybrid Checkpointing

In the quasi-synchronous checkpointing approach, a global checkpoint is similar to the approach of coordinated checkpointing while rollback propagation can be avoided by forcing additional un-coordinated local checkpoint in processes [22]. There are three factors contributing to checkpointing overhead in this approach.

i) Processes are allowed to take their checkpoints asynchronously.
ii) Processes take forced checkpoints on receiving some application message depending upon conditions.
iii) Processes may take checkpoint on receiving checkpoint request message. If a process takes forced checkpoint related to current initiations then it convert the forced checkpoint in to tentative one and if it already takes checkpoints, ignore the checkpoint request.

Quasi-synchronous checkpointing algorithms can be classified into two categories [7].

- *Model based checkpointing*: it relies on preventing patterns of communications and checkpoints that could result in inconsistent states among the existing checkpoints. Here a model is set up to detect the possibility that such patterns could be forming within the system, according to some heuristic and a checkpoint is usually forced to prevent the undesirable pattern from occurring.
- *Index based checkpointing*: Index-based checkpointing works by assigning monotonically increasing indexes to checkpoints, such that the checkpoints having the same index at different processes form a consistent state. The indices are piggybacked on application messages to help receivers decide when they should force a checkpoint.

3) Coordinated/Synchronous/Communication induced Checkpointing

Coordinated checkpointing is a commonly used technique for fault tolerant in mobile distributed systems. In coordinated checkpointing approach all the processes communicate and synchronize through system messages before taking checkpoint and coordinate their checkpointing actions in such a way that checkpointing approach yields a CGS. In some approaches initiator of the checkpointing process forces the dependent processes (minimum processes) In coordinated approach consistent global state is achieved during run-time, while in the independent approach the determination of a consistent recovery line was left to the recovery phase, which could result in some rollback propagation [22].

3.2.4 Comparison between Checkpoint Schemes

Table 1. Comparison between uncoordinated, coordinated and qusi-synchronoun checkpointing

	Uncoordinated Checkpointing	**Coordinated Checkpointing**	**Qusi-Synchronous Checkpointing**
Efficiency	High for small number of MHs	High for large number of MHs	High
Checkpoint/Process	Multiple	Single	Multiple
Domino effect	Possible	No	No
Orphan message	Possible	No	Possible
Scalable	No	Minimal	Not scale for large number of processors
Recovery Cost	High as global consistent state is not predictable	Low as global consistent state is predictable	High
Recovery Complexity	Yes	No	Yes
Rollback	Unbounded	During fault, processes rollback to last committed checkpointed state	Possibly several checkpoints
Overhead	Large storage and log management overhead	Minimum storage overhead and negligible overhead in failure free execution	High latency and memory and disk overhead
Advantages	No need to exchange any control message and save their checkpoint individually	Lower overhead in stable storage, Recovery simple and predictable, not suffer from domino effect.	Preventing domino effect piggybacking and information exchanged by the processes
Disadvantages	Domino effect possibility, storage overhead & complex garbage collection	Synchronization message overhead and large latency for saving checkpoints	Requires high performance parallel processor

Coordinated checkpointing algorithms are made up by using the following scheme:

- *All process checkpointing:* This requires all processes in the system to participate in every checkpointing session.
- *Minimum process checkpointing*: These algorithms only forces those process to take their checkpoints which communicated with the initiator directly or indirectly since the last checkpoint need to take new checkpoints.
- *Blocking:* Blocking algorithms force all relevant processes in the system to block their underlying computation during checkpointing latency.
- *Non-blocking:* In non-blocking algorithms applications processes are not blocked when checkpoints are being taken.

As mobile computing faces many new challenges such as low wireless bandwidth, frequent disconnections and lack of stable storage at mobile nodes. These issues make traditional checkpointing techniques unsuitable to checkpoint mobile distributed systems [1,5,15]. A good checkpoint algorithm for mobile systems needs to have following characteristics [10]. It should impose low memory overheads on MHs and low overheads on wireless channels. The disconnection of MHs should not lead to infinite wait state. The checkpointing algorithm should avoid awakening of an MH in doze mode operation. The algorithm should be non-blocking and minimum-process.

There is a tradeoff between coordinated and uncoordinated checkpointing approach for mobile systems. Some of the approaches advocate coordinated checkpointing[1-4,8,11,13,15,17,21,24-26], as it free from domino-effect and others advocate un-coordinated checkpointing [5], due to lots of synchronization overhead caused by coordinated approach. But un-coordinated checkpointing in true sense is not suitable mobile computing and even for distributed systems due to number of reasons [1]. If the frequency of local checkpointing is high, each process will have multiple checkpoints, which require a large amount of stable storage and introduces a lot of communication overhead in mobile computing systems. The stable storage and communication overheads can be reduced by taking local checkpoints less frequently. However, this will increase the recovery time as greater rollback and reply will be needed. Even though some algorithm were proposed to reduce the number of checkpoints to be saved on the stable storage, to ensure correctness, a process still needs to keep many more checkpoints in uncoordinated checkpointing algorithms. So if we reduce the synchronization overhead from in coordinated approach, then it can become quite effective for mobile systems [27].

In coordinated checkpointing, processes take checkpoints in such a manner that the resulting global state in consistent. Mostly it follows two-phase commit structure [1,2,4,8,13,19,26,27,] [31]. 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 last checkpointed state.

4. PERFORMANCE ANALYSIS OF CHECKPOINTING ALGORITHMS

We analyze and evaluate the checkpointing algorithms on the basic of blocking time, synchronization message overhead, number of processes required to checkpoint, number of useless checkpoint, piggybacked information messages onto computation messages and concurrent executions. We use the following notations:

N_{mss}	Number of MSSs.
N_{mh}	Number of MHs.
C_{st}	Cost of sending a message between any two MSSs.
C_{pp}	Average cost of sending a message between two processes.
C_{wl}	Cost of sending a message from an MH to its local MSS (or vice versa).
C_{bst}	Cost of broadcasting a message over static network
C_{search}	Cost incurred to locate an MH and forward a message to its current local MSS, from a source MSS.
T_{st}	Average message delay in static network.
T_{wl}	Average message delay in the wireless network.
T_{ch}	Average delay to save a checkpoint on the stable storage. It also includes the time to transfer the checkpoint from an MH to its local MSS.
T_{ch_static}	Average delay for a fixed host to save a checkpoint on the stable storage.
T_{search}	Average delay incurred to locate an MH and forward a message to its current local MSS.
N	Total number of processes in the system.
N_{min}	Number of minimum processes required to take checkpoints.
N_{mut}	Number of useless mutable checkpoints [1].
N_{ind}	Number of useless induced checkpoints [2, 19].
N_{dep}	Height of the checkpointing tree in Koo-Toueg [4] algorithm

4.1 All Process Blocking Algorithms

All process blocking algorithm can be differentiates on basic of FIFO and non-FIFO channels. Barigazzi-Strigini[29], Deng-Park[28] and Kim-Park[17] proposed checkpointing algorithm which forces to all processes and block their underlying computation during current checkpointing interval by assuming the FIFO channel. Later, Leu-Bhargwa [30] proposed all blocking algorithm which does not assume that the channel is FIFO. Such algorithms are not suitable for mobile environment. Hence, the problem of minimizing the number of synchronization messages and checkpoints is become a crucial issue in mobile system as wireless network has limited bandwidth and mobile nodes have limited computation, storage and energy conservation requirement. It is mostly desirable that a coordinated checkpoint algorithm forces a minimum number of processes to take checkpoints [14].

4.2 Minimum-process Blocking Algorithms

Minimum- process blocking algorithms has the lowest synchronization overhead in the comparison of all-process blocking algorithms. The algorithms proposed in Koo-Tong[4],Cao-Singhal[13], P.Kumar[26],], Higaki-Taki[27] has the lowest among the blocking algorithms[17], [28]-[30] which try to minimize the number of synchronization messages and the number of checkpoints during checkpointing.

The koo-Toueg[4] proposed a minimum process coordinated checkpointing algorithm for distributed systems with the cost of blocking of processes during checkpointing. However this algorithm requires minimum number of synchronization message and number of checkpoints but each process uses monotonically increasing labels in its outgoing messages. The initiator process sends the checkpoint request to P_i only if it has received m from P_i in the current CI. Similarly, P_i sends the checkpoint request to other processes. In this way, a checkpointing tree is formed and at last the leaf node processes take checkpoints. The time taken to collect coordinated checkpoint in mobile systems may be too large due to mobility, disconnections and unreliable wireless channels. The extensive blocking of processes may degrade the system performance. The blocking time and synchronization message overhead in [4] are $N_{mh} *(4*T_{wl} + T_{ch} + T_{search})$ and $N_{mh} * (C_{wl} + C_{search})$ respectively[Refer Table 2]. Thus extensive blocking of processes may degrade the system performance.

In [13], author proves that there does not exist a non-blocking algorithm that forces only a minimum number of processes to take their checkpoint. Every process maintains direct dependencies in a bit array of length n for n processes. Initiator process collects the direct dependencies and makes a set of interacting processes (S_{forced}), which need to checkpoint along with the initiator. During blocking time, processes can do their normal computations but cannot send any messages. On the basic of this result [13, 26] proposed minimum process blocking algorithm. Both the algorithms [13] and [26] have approximately same blocking time and message overhead. The algorithms [13, 26] block during the time, when MSS sends the dependency vector to its local dependent and receives the checkpoint request. Therefore, the blocking period of both the algorithms in worst case is $2T_{st}$.The coordination message overhead in worst case including the following: (a) $2C_{wl}$ – the checkpoint request message from initiator process to its local MSS and it's acknowledged. (b) $3C_{bst}$: the initiator MSS broadcast send dependency, take checkpoint and commit message to all MSSs. (c) $2N_{mss} * C_{st}$: MSSs send dependency vector to their processes and receive acknowledge. (d) 3 $N_{mh} * C_{wl}$: MSSs send checkpoint request, commit requests to relevant processes and receive acknowledge. So total message overhead (say TMO_{minp}) in worst case in [13,26] is $3_{broad} + 2C_{wl} + 2N_{mss}*C_{st} + 3 N_{mh} *C_{wl}$. As shown in Table 2, algorithms [13, 26] avoids the search cost and dramatically reduces the blocking time from $N_{mh} *(4*T_{wl} + T_{ch} + T_{search}$ to $2*T_{st}$ and cuts the message overhead cost by half compared to Koo-Toueg algorithm. Hence these algorithms avoid the search cost and minimize the number of checkpoints during checkpointing.

Higaki and Takizawa [27] have shown that there is a high probability that at least one MH will fail to record the checkpoint synchronously with other nodes and thus will render whole checkpointing effort useless. Such successive unsuccessful efforts will waste the scarce resources of mobile systems and will not allow the normal computation to proceed. They proposed a checkpointing protocol where mobile hosts checkpoint independently and fixed ones synchronously and requires blocking of processes during checkpointing. An MSS logs the messages of the MHs in its cell. If an MH fails to take its checkpoint and transfer it to the current MSS, it can try later. MSSs take checkpoints synchronously. A process on an MH can recover independently. When a process on an MH crashes, a new process is created using checkpoint of the crashed MH, and then the logged messages are replayed in the order they were originally received. When a process on an MSS fails, all processes rollback to recent synchronous checkpoint. An MH uses its recent committed checkpoint and message logs to reach to a state consistent with the synchronous checkpoint. The algorithm does not awaken an MH in doze mode operation. This algorithm suffers from the overhead of message logging for MHs. The blocking time and synchronization message overhead in [27] are $2*T_{st} + T_{ch\_static}$ and $2*C_{bst} + N_{mss}*C_{st}$ respectively. As shown in table 2, as compared to Cao-Singhal[13], Higaki-Taki[27] have some higher blocking time but reduced message overheads and compared to Koo-Toueg[4], algorithm have reduced blocking and message overhead.

Table 2. Performance analysis of all-process non-blocking and minimum-process blocking algorithms for distributed and mobile systems.

Analysis Parameters	**All Process but Non-Blocking Algo.**		**Minimum-Process but Blocking Algorithms**			
	Elnozahy et al [8]	S.Neogy et al.[25]	Cao-Singhal [13]	P.Kumar [26]	Koo-Toueg [4]	Higaki-Takizawa [27]
Blocking Time	0	0	$2*T_{st}$	$2*T_{st}$	$N_{dep} *T_{ch}$	$2*Tst+ T_{ch\_static}$
Number of checkpoints	N	$N+1$	N_{min}	N_{min}	N_{min}	N_{mss}
Message Overhead	$2*C_{bst}+ N *C_{pp}$	$2*C_{bst} + N*C_{pp}$	$3C_{bst}+2C_{wl}+2 N_{mss}* C_{st}+3N_{mh}* C_{wl}$	TMO_{minp}	$N_{mh}*(6C_{wl}+ C_{search})$	$2*C_{bst} +N_{mss}* C_{st}$
Piggybacked Information	integer	Integer	Nil	Integer	integer	Nil
Distributed/ Centralised	Centralis -ed	Centralis -ed	Distributed	Distribut -ed	Distribut -ed	Distribut -ed

4.3 All Process Non-blocking

All the above coordinated checkpointing algorithms [4][13] [17][26]-[30] requires processes to be blocked during checkpointing. During checkpointing, information related to process like all variables, the environment, control information and register values are stored on the stable storage. So, it consume a lot of time which me be long. Therefore, blocking algorithms may reduce the performance of the system [8]. Further to remove blocking overhead, recently non-blocking distributed checkpointing algorithms [6],[8],[21],[25] have received consideration attention.

The Chandy-Lamport [6] algorithm is the earliest all-process non-blocking coordinated checkpointing algorithm. In this algorithm, the global state is constructed by coordinating all processors and logging the channels states at the time of checkpointing. A special messages called are used for coordination and for identifying the messages originating at different checkpoint intervals. This leads to a message complexity of $O(N^2)$ The algorithm is initiated by a centralized node and requires FIFO channels.

The Elnozahy et al.[8] and Neogy-Sinha[25] also design an all-process non-blocking checkpointing algorithm. In these algorithms the initiator broadcast the checkpoint request to all processes the overhead of which is C_{bst}. The initiator receives reply from the N processes the overhead of which is $N*C_{pp}$. At last the initiator broadcasts a commit request to all processes to convert their tentative checkpoints to permanent one. In such way we get the consistent global state with the total message overhead of $(2*C_{bst} + N*C_{pp})$ [Refer Table 2].

Algorithm proposed by Silva and Silva [21] uses a similar idea as [8] except that the processes which did not communicate with others during the previous checkpoint interval do not need to take new checkpoints.

Table 3. Performance Analysis of Minimum-processes non-blocking and minimum-processes non-blocking with useless checkpoints for distributed and mobile systems.

Analysis Parimeter	**Minimum-Process and Non-blocking**		**Minimum-Process and Non-blocking with Useless Checkpoints**		
	S.K.Gupta et al.[27]	B.Gupta et al.[31]	Cao_ Singhal [1]	P.Kumar et al.[2]	Lalit Kumar et al. [19]
Blocking Time	0	0	0	0	0
Number of checkpoints	N_{min}	N_{min}	$N_{min}+N_{mut}$	$N_{min}+N_{ind}$	$N_{min}+N_{ind}$
Message Overhead	$2*N_{min}*C_{pp} + C_{bst}$	C_{bst}	$2*N_{min}*C_{pp} + \min(N_{min}*C_{pp}, C_{bst})$	$3*C_{bst} + 2*N_{min}*C_{pp}$	$3C_{bst}+2C_{wl}+2N_{mss}*C_{st}+3N_{mh}*C_{wl}$
Piggybacked Information	Integer	Nil	Integer	Integer	integer
Concurrent Execution	No	No	Yes	No	No
Useless Checkpoint	nil	nil	Present	Present	Present
Single Phase/ Two Phase	Two	Single	Two	Two	Two
Non-deterministic	No	Yes	Yes	Yes	Yes

Therefore, algorithms [6][8][21][25] suffer from the disadvantages of centralized algorithms, such as one-site failure, traffic bottle-neck, etc. and there is no easy way to make it distributed without significantly increasing message overhead[1]. Moreover their algorithms require almost all processes to take checkpoints, even though many of them are unnecessary. Since some processes may be in the doze mode, broadcast may waste their energy and processor power.

However, algorithms proposed in [6,8,21,25] dramatically increases the performance of the system in the comparison of blocking algorithms [4][13] [17][26]-[30] which requires processes to be blocked during checkpointing. As checkpointing time may be long and can reduce the performance of system. But algorithms [6,8,21,25], forces to all processes in the system to take their checkpoints for each checkpoint initiation, even though many of them may not be necessary as [17], [28]-[30].This may waste the energy and processor power of the processes which are in doze mode. We compare only Elnozahy et al. [8] and Neogy-Sinha[25] with minimum process blocking algorithms, as these has lowest overhead in among algorithms[6,8,21,25].

4.4 Non-blocking Minimum-process

A good checkpointing protocol for mobile distributed systems should have low overheads on MHs and wireless channels and should avoid awakening of MHs in doze mode operation. The algorithm should be non-intrusive and should force minimum number of processes to take their local checkpoints [15].

All the above algorithms either minimize the number of synchronization messages and the number of checkpoints [17], [28]-[30] or make checkpointing algorithm non-blocking [6][8] [21],[25].

Prakash-Singhal proposed an algorithm in [15] by combing both minimum-process and non-blocking approach. This algorithm only forces the minimum number of processes to take checkpoints without blocking of the underlying computation. Cao and Singhal [13] have shown that the algorithm [15] can leads to inconsistencies. The authors also proved that there does not exist a non-blocking algorithm which forces only minimum number of processes to take checkpoints. Due to the inconsistency in algorithm [15], we do not compare it with other algorithms.

In [27], S.K. Gupta et al. proposed a minimum process non blocking checkpointing algorithm for deterministic mobile distributed systems with a message overhead of $2*N_{min}*C_{pp}+C_{bst}$ [Refer Table 3]. In deterministic system, if two processes starts in the same state, and both receive the identical sequence of inputs, they will produce the identical sequence outputs and will finish in the same state. In such case state of a process is completely determined by its starting state and by sequence of messages it has received [27].

B.Gupta et al.[31] presented a single phase an efficient non-blocking coordinated checkpointing algorithm to determine the global consistent state. The algorithm produce $2*N_{min}*C_{pp} + C_{bst}$ [Refer Table 3] message overhead and forces the minimum number of processes. Simulation result have shown that algorithm requires much less number of control (system) message in the comparison of [4,8, 25].

4.5 Minimum-process Non-blocking with Useless Checkpoints

A good coordinated checkpointing algorithm for mobile distributed system should be non-intrusive and force minimum number of processes to take their local checkpoints [15] but in algorithm [13] Cao-Singhal proved that non-intrusive and minimum process algorithms does not exist in coordinated system**.** The algorithms [1][2][19] achieved non-intrusiveness with minimum-process in coordinated systems by using some useless checkpoints.

In [1], Cao-Singhal proposed a mutable checkpoint based non-blocking minimum-process coordinated checkpointing algorithm. This algorithm completes its processing in the following three steps. First initiator MSS sends tentative checkpoint request to minimum number of processes that need to take checkpoint. The synchronization message overhead for this is $N_{min} *C_{pp}$. Secondly MSS_{ini} gets the acknowledgement from all processes to whom it sent checkpoint request. Hence message overhead $2* N_{min} *C_{pp}$ is needed in first two phases. At last MSS_{ini} sends the commit request to convert its tentative checkpoint into permanent. In this case it takes $min(N_{min}* C_{st}, C_{bst})$. Hence algorithm [1] determine consistent global state with the message overhead cost $2* N_{min} * C_{pp} + min(N_{min}* C_{pp}, C_{bst})$ and average number of checkpoints $N_{min}+ N_{mut}$ [Refer Table 3]. Mutable checkpoints taken as: If any process sends a computation message to another process after receiving the checkpoint request, the receiving process first take the mutable checkpoint first and then process the message. Later, this mutable checkpoint converted to tentative, if it receives checkpoint request related to the current initiation; otherwise it become the useless checkpoint. The number of useless checkpoints in [1] may be exceedingly high in some situations [19].

P. Kumar et al. [2] proposed five phase minimum-process non-intrusive coordinated checkpointing algorithms to reduce the height of the checkpointing tree and the number of useless checkpoints. It follows the following steps in a distributed system which has (n+1) processes. (i) Initiator process broadcasts the dependency vector request to all processes. (ii) Receives the dependency vector from all processes and then initiator process compute minimum set of processes which are directly or transitively dependent on the initiator process. (iii) Take own tentative checkpoint and send the tentative checkpoint request to the processes which belongs to the minimum set.(iv) Initiator process receives the responses of taking tentative checkpoint (v) initiator process send the commit or abort message to all the processes. The synchronization message overhead to complete the checkpointing process using algorithm[2] is given as $3*C_{bst} + 2*N_{min} * C_{pp}$.[Table 3] Here $3C_{bst}$ is the total cost of broadcasting sends ddv[](C_{bst}, take tentative checkpoint request(C_{bst}) and commit(C_{bst}) messages to all MSSs by the initiator MSS. $2*N_{min}*C_{pp}$ is the total cost of sending checkpoint request message to the minimum number of processes that need to take checkpoints($N_{min}*C_{pp}$) and reply to the initiator after taking the tentative checkpoint($N_{min}*C_{pp}$). Hence algorithm [2] determines the global consistent state by using $N_{min}+ N_{indu}$ average number of checkpoint and $3*C_{bst} + 2*N_{min} * C_{pp}$ message overhead cost but our proposed algorithm by using N_{min} and $3* N_{min} * C_{pp}$ respectively [Refer Table 3]. Thus algorithm [2] takes less useless checkpoint in the comparison of [1] but suffer from the overhead of collecting dependencies, computing the minimum set and broadcasting the minimum set on the static network.

L. Kumar et al. [19] also proposed minimum-process non-intrusive coordinated checkpointing algorithms, where number of useless checkpoints is reduced as compared to [1][2]. In both of these algorithms, initiator MSS collects the direct dependency vectors of all processes, computes the minimum set, and sends the checkpoint request along with the minimum set to all MSSs. In algorithm [19], during the time P_i sends its direct dependency vectors and receives the minimum set, if P_i processes m which changes its own direct dependency vector, P_i takes induced checkpoint before processing m. In this way, fresh dependencies, created during checkpointing, do not alter the computed minimum set. The proposed minimum process non-instrusive approach have the total message overhead cost is $3C_{bst}+2C_{wl}+2N_{mss}* C_{st} +3N_{mh}* C_{wl}$ with $N_{min} +N_{ind}$ checkpoints [Refer Table 3]. Simulations results have shown that the number of useless checkpoints in [19] is negligible as compared to [1]. But algorithm [19], is also suffer from the overhead as in [2].

5. CONCURRENT INITIATIONS

Most of the proposed checkpointing algorithms not addressing the multiple concurrent initiations in their algorithms, as it may exhaust the limited battery and congest the wireless channels. The authors claim in [1],[3] that their algorithm supports concurrent initiations. But in[20] authors proves that the algorithm in[1],[3] are designed to only handle the situation where the system has only one checkpoint initiator at a time and can cause inconsistency when there are multiple forced checkpoints or multiple concurrent checkpoint initiations. In[20] author also point out the reasons for inconsistency in [1],[3] and proposed a consistent checkpointing algorithm which supports concurrent executions. In[26], the author point out following problems in allowing concurrent initiations in minimum-process checkpointing protocols, particularly in case of mobile distributed systems:

i) If P_i and P_j concurrently initiate checkpointing and P_j belongs to the minimum set of Pi, then P_j's initiation will be redundant one. Some processes, in P_j's minimum set, will unnecessarily take multiple checkpoints by hardly advancing their recovery line. In other words, an MH may be asked to store multiple checkpoints in its local disk. It may also transfer multiple checkpoints to its local MSS.
ii) Sometimes, multiple triggers need to be piggybacked onto normal messages. Trigger contains the initiator process identification and its csn. Even if a process takes a checkpoint and no concurrent initiation is going on, it will piggyback its trigger, unnecessarily. If we do not allow concurrent initiation, no trigger is required to be piggybacked onto normal messages. Hence, concurrent initiations increase message size.

6. COMPARATIVE STUDY

Barigazzi et a. in [3] proposed first coordinated checkpointing algorithm by assuming that communication between processes are atomic, which is too restrictive.

In [17],[29] proposed FIFO channel based checkpointing algorithm which forces to all process to take checkpoint and blocks their computation during checkpointing by relaxing the atomic assumption. Later, Leu-Bhargava[30] proposed coordinated checkpointing algorithm as [17],[29] by relaxing the FIFO channel assumption. However, algorithm [30] does not consider lost messages in checkpointing and recovery. Deng-Park [28] proposed an algorithm to deal with orphan and lost messages efficiently. Algorithms[4],[13],[28] has the lowest overhead among the blocking algorithms [17],[28]-[30], which try to minimize the number of synchronization messages and the number of checkpoints during checkpointing by forcing the process which are directly or transitively dependent upon initiator process since the last checkpoint. Koo-Tong [4] have high blocking time and double message overhead cost in the comparison of [13,26][Refer Table 2]. As shown in table 2, as compared to Cao-Singhal[13], Higaki-Taki[27] have some higher blocking time but reduced message overheads cost and compared to Koo-Toueg[4], algorithm have reduced blocking and message overhead. However in [4],[13],[28], if any one of the involved process reply negatively, the entire process is aborted. Kim-Park [17] and Higaki-Taki[27] proposed an improved scheme that allow the new checkpoints in some subtrees to be committed, while others are aborted. Further to reduce system messages, time based checkpoints are used in algorithm [32].

All the above coordinated checkpointing algorithms are forces all- process or minimum-process [4][13] [17][26]-[30] to take checkpoints during current checkpointing interval but requires blocking of processes during checkpointing. Since saving checkpoints takes a long time which increase blocking time and it may reduce the performance of system [8]. In [6], Chandy-Lamport presented first non-blocking checkpointing algorithm for coordinated checkpointing which forces to all processes to take their checkpoints, however many of them may not be necessary. Further, Elnozahy et al.[8] an Neogy-Sinha[25] also design non-blocking checkpointing algorithms which forces to all process to take checkpoints. Silva-Silva [21] uses a similar idea as[8] except that the processes which did not communicate with others during the previous checkpoint interval do not need to take new checkpoints. Therefore, algorithms[6],[8][21][25] suffer from the disadvantages of centralized algorithms, such as one-site, traffic bottle-neck, etc. and there is no easy way to make it distributed without significantly increasing message overheads[1].

All the above algorithms either minimize the number of synchronization messages and number of checkpoints [17],[28]-[30] or make it non-blocking[6][8][21][25]. Prakash-Singhal proposed an algorithm in [15] by combing both minimum-process and non-blocking approach. This algorithm only forces the minimum number of processes to take checkpoints without blocking of the underlying computation. Cao and Singhal [13] have shown that the algorithm [15] can leads to inconsistencies. The authors also proved that there does not exist a non-blocking algorithm which forces only minimum number of processes to take checkpoints. S.K.Gupta et al. [27] proposed a minimum process non-blocking checkpointing algorithm for non-deterministic mobile distributed systems. B.Gupta et al.[31] presented a single phase non-blocking coordinated checkpointing algorithm to determine the global consistent state.

A good coordinated checkpointing algorithm for mobile distributed system should be non-intrusive and force minimum number of processes to take their local checkpoints [15] but in algorithm [13] Cao-Singhal proved that non-intrusive and minimum process algorithms does not exist in coordinated system**.** The algorithms [1][2][19] achieved non-intrusiveness with minimum-process in coordinated systems by using some useless checkpoints.

Cao-Singhal[1] proposed a mutable checkpoint based non-blocking minimum-process coordinated checkpointing algorithm. The number of useless checkpoints in [1] may be exceedingly high in some situations mention in algorithm [19]. Author also proved that concurrent executions is allowed in his algorithm [1], but in algorithm [20] author prove that algorithm [1] may lead to inconsistency during concurrent execution. P.Kumar et al. [2] takes less useless checkpoint in the comparison of [1] but suffer from the overhead of collecting dependencies, computing the minimum set and broadcasting the minimum set on the static network. L. Kumar et al. [19] also proposed minimum-process non-intrusive coordinated checkpointing algorithms, where number of useless checkpoints is reduced as compared to [1][2]. Simulations results have shown that the number of useless checkpoints in [19] is negligible as compared to [1]. But algorithm [19], is also suffer from the overhead as in [2].

7. PROPOSED CHECKPOINTING ALGORITHM

In this paper we proposed non-blocking coordinated checkpointing algorithm for distributed mobile systems with lesser number of checkpoints. Our checkpoint algorithm does not have any synchronization message overhead and use time to indirectly coordinate to create the consistent cut in distributed mobile system.

With the new proposed checkpointing algorithm, each process checks the status of checkpointing periodically. At the time of expiry of local timer, before taking a checkpoint, each process check, if there is a forced checkpoint in the current checkpoint interval. It there exist a forced checkpoint, then the content of forced checkpoint, which is stored on local memory of MH, is written to a stable storage of its local MSS. After checkpointing a sequence number is increased by one and checkpointing time is updated. To reduce the number of checkpoint, our proposed algorithm does not take checkpoint if it does not send any message during its current checkpoint interval.

During normal operation to remove the domino effect and inconsistency, each process P_i sends the checkpoint sequence number (csn) of P_i and Last_csn$_i$[j] with the message. When P_j receives the message and observe that csn$_i$[j] and csn$_i$ is equal, it is observable that a domino cycle is going to happen. So it forces to take a forced checkpoint to break the cycle. Each process before receiving the message compares its csn with the csn number of the sender that is logged with each message.

7.1 Data Structure

m: Computation message sent by a process.

Timer$_i$: time of the process $P_{i.}$

csn$_i$: A checkpoint sequence number for the current checkpoint interval of P_i. it is initially set to zero and incremented by 1 each time process takes new checkpoint. Each message sent from process P_i to P_j piggybacked with the current csn$_i$.

csn$_i$[j]: An integer vector which denotes the csn of P_j currently known by P_i.

Last_csn$_i$[j]: An integer array which denotes the csn$_j$ carried in the last message from P_j before the latest checkpoint was taken.

send$_i$: A boolean variable with default value 0 and set to 1 if P_i sent a message in its current checkpointing interval.

FV: A boolean variable with default value 0 and set to 1 if P_i takes a its forced checkpoint during its current checkpointing interval.

7.2 Checkpointing Algorithm

```
1. At each process P_i in distributed system (1≤i≤n) set its local timer_i.
2. On sending and receiving message:
   When P_i sends a message to P_j
         Send (m, csn_i, last_csn_i[j],)
    When P_j receves(m, csn_i, Last_csn_i[j] from P_j )
         if (csn_i = Last_csn_j[i])
```

```
            { Take forced checkpoint;
              Set FV=0; Last_csn$_i$=csn$_i$ ;  csn++; csn$_i$[j] = csn$_{j;}$
                Receive message m;
            }
3. if timer$_i$ expires
        if (FV =0 &&  Send$_i$ = =0)// not any forced checkpoint taken
          { do not take checkpoint and continue normal execution;}
        else if(FV =0 && Send$_i$ = = 1)
           { Take checkpoint; set send$_i$= 0; csn$_i$ ++; resume continue execution;}
        else // FV= =1
          {  Convert forced checkpoint in permanent checkpoint; csn$_i$++; set FV=0;}
```

7.3 Handling Node Mobility and Disconnections

An MH may be disconnected from the network for an arbitrary period of time. Disconnection is distinct from failure. Disconnections are elective by nature. An MH informs its MSS prior to its disconnection. Abrupt or non-volunteer disconnection is termed as a failure. The Checkpointing algorithm may generate a request for such MH to take a checkpoint. Delaying a response may significantly increase the completion time of the checkpointing algorithm.

When an MH, say MH_i, disconnects from an MSS, say MSS_k, MH_i takes its own checkpoint, say *disconnect_ckpt*$_i$, and transfers it to MSS_k. MSS_k stores all the relevant data structures and *disconnect_ckpt*$_i$ of MH_i on stable storage. During disconnection period, MSS_k acts on behalf of MH_i as follows. In minimum-process checkpointing, if MH_i is in the minimum set, say *minset[]*, *disconnect_ckpt*$_i$ is considered as MH_i's checkpoint for the current initiation. In all-process checkpointing, if MH_i's *disconnect_ckpt*$_i$ is already converted into permanent one, then the committed checkpoint is considered as the checkpoint for the current initiation; otherwise, *disconnect_ckpt*$_i$ is considered. On global checkpoint commit, MSS_k also updates MH_i's data structures, e.g., *ddv*[] (direct dependency vector). On the receipt of messages for MH_i, MSS_k does not update MH_i's *ddv*[] but maintains them in a queue, say *message_q*.

When MH_i, enters in the cell of MSS_j, it is connected to the MSS_j. Before connection, MSS_j collects MH_i's *ddv*[], and other information from MSS_k; and MSS_k discards MH_i's support information and *disconnect_ckpt*$_i$. MSS_j sends the messages in *message_q* to MH_i.

7.4 Handling Failures during checkpointing

Since MHs are prone to failure, an MH may fail during checkpointing process. Sudden or abrupt disconnection of an MH is also termed as a fault. If the failed process is not required to checkpoint in the current initiation or the failed process has already taken its tentative checkpoint, the checkpointing process can be completed uninterruptedly. If the failed process is not the initiator, one way to deal with the failure is to discard the whole checkpointing process similar to the approach in [4], [9]. The process detecting the failure informs the initiator, which aborts current checkpointing process. If the failed process is a checkpoint initiator, and the failure occurred before the process sent out *commit* or *abort* messages, on restarting after failure, it aborts the checkpointing activity corresponding to its initiation. If the initiator fails after sending *commit* or *abort* message, it has nothing to do for the current initiation.

The above approach seems to be inefficient, because, the whole checkpointing process is discarded even when only one participating process fails. The more efficient technique has been proposed by Kim and Park [17]. In the approach, a process commits its tentative checkpoints if none of the processes, on which it transitively depends, fails; and the consistent recovery line is advanced for those processes that committed their checkpoints. The initiator and other processes which transitively depend on the failed process have to abort their tentative checkpoints. Thus, in case of a node failure during checkpointing, total abort of the checkpointing is avoided.

8. CONCLUSION

The mobile wireless networks present challenges in designing fault-tolerant systems because of the host mobility, limited bandwidth on wireless link, limited local storage on MH, limited battery power etc. Several checkpointing approaches have been proposed for distributed mobile systems. In coordinated checkpointing processes coordinate through system message before taking checkpoints. These synchronization messages contribute to extra overhead. Coordinated checkpointing algorithms may be minimum-process, blocking, non-blocking and takes some useless checkpoints to determine the consistent global state. In this paper we analyzed and compare different coordinated checkpointing algorithms for distributed mobile systems on the basic of blocking time, synchronization message overhead, number of processes required to checkpoint, number of useless checkpoint, piggybacked information messages onto computation messages and concurrent execution. Every checkpointing algorithms try to minimize the checkpointing overhead but not fit in mobile environment from all perspective. An efficient checkpointing algorithm for mobile distributed systems should have the following desirable feature. These features will ensure efficient use of wireless channel bandwidth and conserve the energy of MH battery.

- In coordinated checkpointing processes coordinated through message passing. These messages contribute extra overhead. So minimizing the synchronization messages when checkpointing is in progress become a crucial issue in mobile system due to the low bandwidth constraint and energy conservation requirement.
- Checkpoint algorithm should have minimum checkpoint latency from the time a process initiate a checkpoint request to the time the global checkpointing process complete.
- Checkpointing algorithm should have low overhead on MHs and wireless channels and should avoid awakening of MHs in doze mode operation.
- The checkpointing process should be non-blocking
- The checkpointing should forces minimum number of processes to take checkpoint.
- The algorithm should be domino-effect free.
- The disconnection of MHs should not lead to infinite wait state.
- The algorithm should not have useless checkpoints. However in case of minimum-process coordinated checkpointing algorithms some useless checkpoints can be taken.

REFERENCES

[1] 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.

[2] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta, A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems, *Proceedings of IEEE ICPWC-2005*, January 2005.

[3] 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.

[4] R.Koo and S.Toueg. Checkpointing and Roll-back Recovery for Distributed systems, *IEEE Transactions on Software Engineering*, pages 23-31, January 1987.

[5] 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.

[6] 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.

[7] 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.

[8] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., The Performance of Consistent Checkpointing, *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pp. 39-47, October 1992.

[9] Y.Dang,E.K. Park, Checkpointing and Rollback-Recovery Algorithms in Distributed Systems, *J. Systems and Software*, pp. 59-71, Apr. 1994.

[10] Hung, S. T., Detecting Termination of Distributed Computations by External Agents, *Proceeding 9th Int' Conf. Distributed Computing System*, pp.79-84, 1989.

[11] R. Baldoni, J.M. Helary, A. Mostefaoui and M.Raynal, *A Communication-Induced Checkpoint Protocol that Ensures Rollback-Dependency Tractability*, In Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, 1997, 68-77.

[12] J.M. Helary, A. Mostefaoui and M. Raynal, *Communication-Induced Determination of Consistent Snapshot*, In Proceedings of the 28th International Symposium on Fault-Tolerant-Computing , 1998,208-217.

[13] 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.

[14] Guohui Li and LihChyun Shu, A Low-Latency Checkpointing Scheme for Mobile Computing Systems.

[15] 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.

[16] I. Akyildiz, J. Mcnair, J. Ho, H. Uzunalioglu, and W. Wang, Mobility Management in ext-Generation Wireless Systems *IEEE*, vol. 87, no. 8. pp.. 1347-1384, Aug 1999.

[17] 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.

[18] C. Perkins, Mobile IP, *IEEE Comm. Magazine*, vol. 35, pp. 84-99, May 1997.

[19] L. Kumar, M. Misra, R.C. Joshi, Low overhead optimal checkpointing for mobile distributed systems Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[20] Ni, W., S. Vrbsky and S. Ray, Pitfalls in Distributed Nonblocking Checkpointing, *Journal of Interconnection Networks*, Vol. 1 No. 5, pp. 47-78, March 2004.

[21] L.M. Silva and J.G. Silva, Global Checkpointing for Distributed Programs, Proc. 11th Symp. Reliable Distributed Systems, pp. 155-162, Oct. 1992.

[22] Najib A. Kafahi, Said AI-Bokhitan andAhmed AI-Nazer, On Disk-based and Diskless Checkpointing for Parallel and Distributed Systems: An Empirical Analysis, Information Technology Journal 4(4): 367-376, 2005.

[23] Weigang Ni, Susan V. Vrbsky and Sibabrata Ray, Low-cost Coordinated Checkpointing in Mobile Computing Systems",*Proceeding of the Eighth IEEE International Symposium on Computers and Communications*, 2003.

[24] L.Kumar, P.Kumar, A Synchronous Checkpoiting Protocol for Mobile Distributed System: Probabilistic Approach, *International Journal of Information and Computer Security* 1(3)(2007), 298-314.

[25] S. Neogy, A. Sinha, P.K. Das, A Checkpointing Protocol for Distributed System Processes, TENCON 2004. 2004 IEEE Regions 10 congerence vol.B. no.2,pp 553-556, November 2004, Thailand.

[26] Parveen Kumar, A low-cost hybrid coordinated checkpointing protocol for mobile distributed systems, Journal of Mobile Information Systems, vol 4, Number 1, 2008.
[27] H Higaki & M Takizawa, Checkpoint-recovery Protocol for Reliable Mobile Systems,Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.
[28] Y. Deng and E.K. Park. “Checkpointing and Rollback-Recovery Algorithms in Distributed System”, Journal of Systems and Software, pages 59-71, April 1994.
[29] G. Barigazzi and L. Strigini, “Application-Transparent Setting of Recovery Points”, Digest of Papers Fault-Tolerant Computing Systems-13, pp. 48-55, 1983.
[30] P.Y. Leu and B. Bhargava, “Concurrent Roubst Checkpoinitng and Recovery in Distributed Systems,” Proc. Fourth IEEE Int’l Cong. Data Eng., pp. 154-163, 1988.
[31] Bidyut Gupta, Shahram Rashimi, rishad A. Rias, and Guru.Banglore, A low-Overhead Non-blocking Checkpointing Algorithm for Mobile Computing Environment, LNCS 3947, pp. 597-608, 2006.
[32] P. Ramanathan and K.G. Shin, “Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System”, IEEE Trans. Software Eng., pp. 571-583, June 1993.
[33] Mukesh Singhal, Niranjan G. Shivaratri, “Advance Concept in Operating System ” Tata Mcgraw-Hill, 2005, pp295-368.
[34] Rendell, B., “Reliable Computing Systems,” Operating Systems :An Advanced course, Springer-Verlag, New York, 1979, pp. 282-391.
[35] T.H. Lai and T.H. Yang. “On Distributed Snapshot”, In Information Processing Laters, vol. 25, pp. 153-158, 1987.
[36] P.S.Mandal, K. Mukhopadhyaya, “Performance analysis of different checkpointing and recovery schemes using stochastic model” Journal of Parallel and Distributed Computing, 66(2006) 99-107.
[37] Ajay D.Dshemkalylani and Mukesh Singhal “Distributed Computing: Principals, Algorithms, and systems” Cambrige University Press.

Surender Kumar is Sr. Lecturer in the Department of Information Technology at Haryana College of Technology & Management Kaithal (Haryana) India. He is pursuing his PhD in Computer Science from Kurukshetra University, Kurukshetra and his M.Tech. from the Ch. Devi Lal University, Sirsa(Haryana) INDIA. His research interests include checkpointing in mobile distributed systems, fault tolerance, mobile computing.

Dr. R.K Chauhan is serving as a Chairman, Department of Computer Sc. & application, Kurukshetra University, Kurukshetra. He has contributed many technical papers in areas including Database, Data Mining & Warehousing, Mobile Computing, Ad-hoc Networks and Checkpointing Algorithms.

Dr. Parveen Kumar received his Ph.D degree in Computer Sc. from Kurukshetra University, Kurukshetra, India, in 2006. He has contributed over 20 technical papers in areas including Checkpointing Algorithms, Mobile Computing and Distributed Systems. He is serving in MIET, Meerut(U.P), INDIA as a Professor in Computer Sc. & Engineering department.