An Analysis of Checkpointing Algorithms for Distributed Mobile Systems

Ajay Khunteta[1], Praveen Kumar[2]

[1]Singhaniya University
Pacheri, Rajasthan, India
Email: ajay_khunteta@rediffmail.com

[2] Department of Computer Science & Engineering
Meerut Institute of Engineering & Technology, Meerut, India, Pin-125005

***Abstract*:** Distributed snapshots are an important building block for distributed systems, and are useful for constructing efficient checkpointing protocols, among other uses. Direct application of these algorithms to mobile systems is not feasible, however, due to differences in the environment in which mobile systems operate, relative to general distributed systems. The mobile computing environment introduces new challenges in the area of fault-tolerant computing. Compared to traditional distributed environments, wireless networks are typically slower, providing lower throughput and latency, comparing to wireline networks. In addition, the mobile hosts have limited computation resources, are often exposed to harsh operating environment that makes them more likely to fail, and can roam while operating. Over the past two decades, intensive research work has been carried out on providing efficient checkpointing protocols in traditional distributed computing. Recently, more attention has been paid to providing checkpointing protocols for mobile systems. Some of these protocols have been adapted from the traditional distributed environment; others have been created from scratch for mobile systems. Checkpoint is defined as a designated place in a program at which normal processing is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. Checkpointing is the process of saving the status information. This paper surveys the algorithms which have been reported in the literature for checkpointing in Mobile Distributed systems.

***Keywords*:** Checkpointing, rollback recovery, fault tolerant systems, Mobile host, Mobile support station etc**.**

1. Introduction

A distributed system consists of several processes that execute on geographically dispersed computers and collaborate via message -passing with each other to achieve a common goal. In a traditional distributed system all hosts are stationary. Recent advances in portable computers with wireless communication interfaces and satellite services have made it possible for mobile users (mobile computers) to perform distributed applications and to access information anywhere and at anytime. This new computing environment where some hosts are mobile computers connected by wireless communication networks and some are stationary computers connected by a fixed network is called a distributed mobile computing environment. Thus, a distributed mobile system can be considered as a special kind of general distributed systems where some of its hosts are not fixed in their location. This new paradigm is distributed mobile computing. Clearly, a mobile system is not necessarily a distributed system, and mobile computing is not necessarily distributed computing.

A distributed mobile system is characterized by the mobility and poor resource of mobile hosts. These two distinct features raise various new issues and constraints not faced in a stationary distributed system [16], [15]. When designing a protocol involving mobile hosts, there are some issues which have to be taken consideration like limited and vulnerable mobile host local storage, low bandwidth and high channel contention and voluntary disconnection/connection, location cost of mobile help station and energy consumption. All these issues and challenges have made those algorithms devised for traditional distributed system not applicable.

A large class of important problems in distributed systems can be cast as periodically calculating consistent global states and executing some reactions based on the global state that have been taken. This paradigm requires consistently recording the global state of a distributed computing. A global state is a collection of the local states, one from each process of the computation, recorded by a process. The global state is said to be consistent if it looks to all the processes as if it were taken at the same instant everywhere in the system.

There have been many papers on finding consistent global states of a distributed application [17]. However, the constraints imposed by the mobility and poor resource of mobile hosts as outlined above complicate the design of distributed algorithms and applications, and make them inappropriate for distributed mobile computing environments. Besides its use to recover from failures, checkpointing is also used in debugging distributed programs and migrating processes in a multiprocessor system. In debugging distributed programs state changes of a process during execution are monitored at various time instances. Checkpoints assist in such monitoring. Checkpointing a process periodically provides the information necessary to move it from one processor to another. The main objective of this paper is to survey checkpointing algorithms used in mobile distributed systems.

2. Aspects of Checkpointing

Upon a failure, checkpoint-based rollback recovery restores the system state to the most recent consistent set of checkpoints, i.e. the recovery line [Randell 1975]. It does not rely on the PWD assumption, and so does not need to detect, log, or replay nondeterministic events. Checkpoint-based protocols are therefore less restrictive and simpler to implement than log-based rollback recovery. But checkpoint based rollback recovery does not guarantee that pre failure execution can be deterministically regenerated after a rollback. Therefore, checkpoint-based rollback recovery is ill suited for applications that require frequent interactions with the outside world, since such interactions require that the observable behavior of the system through failures and recoveries be the same as during a failure-free execution. Checkpoint-based rollback-recovery techniques can be classified into three categories: uncoordinated checkpointing, coordinated checkpointing, and communication-induced checkpointing.

2.1 Uncoordinated Checkpointing:

Uncoordinated checkpointing allows each process the maximum autonomy in deciding when to take checkpoints. The main advantage of this autonomy is that each process may take a checkpoint when it is most convenient. For example, a process may reduce the overhead by taking checkpoints when the amount of state information to be saved is small [Wang 1993].
But there are several disadvantages. First, there is the possibility of the domino effect, which may cause the loss of a large amount of useful work, possibly all the way back to the beginning of the computation. Second, a process may take a useless checkpoint that will never be part of a global consistent state. Useless checkpoints are undesirable because they incur overhead and do not contribute to advancing the recovery line. Third, uncoordinated checkpointing forces each process to maintain multiple checkpoints, and to invoke periodically a garbage collection algorithm to reclaim the checkpoints that are no longer useful. Fourth, it is not suitable for applications with frequent output commits because these require global coordination to compute the recovery line, negating much of the advantage of autonomy. In order to determine a consistent global checkpoint during recovery, the processes record the dependencies among their checkpoints during failure-free operation [Bhargava and Lian, 1988].

2.2 Coordinated Checkpointing

Coordinated checkpointing requires processes to orchestrate their checkpoints in order to form a consistent global state. Coordinated checkpointing simplifies recovery and is not susceptible to the domino effect, since every process always restarts from its most recent checkpoint. Also, coordinated checkpointing requires each process to maintain only one permanent checkpoint on stable storage, reducing storage overhead and eliminating the need for garbage collection. Its main disadvantage, however, is the large latency involved in committing output, since a global checkpoint is needed before messages can be sent to out side world. A straightforward approach to coordinated check- pointing is to block communications while the checkpointing protocol executes [Tamir and Sequin 1984]. A coordinator takes a checkpoint and broadcasts a request message to all processes, asking them to take a checkpoint. When a process receives this message, it stops its execution, flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgment message back to the coordinator. After the coordinator receives acknowledgments from all processes, it broadcasts a commit message that completes the two-phase checkpointing protocol. After receiving the commit message, each process removes the old permanent checkpoint and atomically makes the tentative checkpoint permanent.
The process is then free to resume execution and exchange messages with other processes. This straightforward approach, however, can result in large overhead, and therefore non-blocking checkpointing schemes are preferable [Elnozahy et al. 1992].

2.3 Non-blocking Checkpoint Coordination

A fundamental problem in coordinated checkpointing is to prevent a process from receiving application messages that could make the checkpoint inconsistent. If channels are FIFO, this problem can be avoided by preceding the first post-checkpoint message on each channel by a checkpoint request, and forcing each process to take a checkpoint upon receiving the first checkpoint-request message. An example of a nonblocking checkpoint coordination protocol using this idea is the distributed snapshot [Chandy and Lamport 1985], in which markers play the role of the checkpoint- request messages. In this protocol, the initiator takes a checkpoint and broadcasts a marker (a checkpoint request) to all processes. Each process takes a checkpoint upon receiving the first marker and rebroadcasts the marker to all processes before sending any application message. The protocol works assuming the channels are reliable and FIFO. If the channels are non-FIFO, the marker can be piggybacked on every post-checkpoint message. Alternatively, checkpoint indices can serve the same role as markers, where a checkpoint is triggered when the receiver's local checkpoint index is lower than the piggybacked checkpoint index [Elnozahy, et al. 1992; Silva 1997].

2.4 Checkpointing with Synchronized Clocks

Loosely synchronized clocks can facilitate checkpoint coordination [Cristian and Jahanian 1991; Tong et al. 1992]. More specifically, loosely synchronized clocks can trigger the local checkpointing actions of all participating processes at approximately the same time without a checkpoint initiator [Cristian and Jahanian 1991]. A process takes a checkpoint and waits for a period that equals the sum of the maximum deviation between clocks and the maximum time to detect a failure in another process in the system. The process can be assured that all checkpoints belonging to the same coordination session have been taken without the need of exchanging any messages. If a failure occurs, it is detected within the specified time and the protocol is aborted.

2.5 Minimal Checkpoint Coordination

Coordinated checkpointing requires all processes to participate in every checkpoint. This requirement generates valid concerns about its scalability. It is desirable to reduce the number of processes involved in a coordinated checkpointing session. This can be done since only those processes that have communicated with the checkpoint initiator either directly or indirectly since the last checkpoint need to take new checkpoints. The following two-phase protocol achieves minimal checkpoint coordination. During the first phase, the checkpoint initiator identifies all processes with which it has communicated since the last checkpoint and sends them a request. Upon receiving the request, each process in turn identifies all processes it has communicated with since the last checkpoints and sends them a request, and so on, until no more processes can be identified. During the second phase, all processes identified in the first phase take a checkpoint. The result is a consistent checkpoint that involves only the participating processes. In this protocol, after a process takes a checkpoint, it cannot send any message until the second phase terminates successfully, although receiving a message after the checkpoint has been taken is allowable.

2.6 Communication-induced Checkpointing

Communication-induced checkpointing avoids the domino effect while allowing processes to take some of their checkpoints independently [14]. However, process independence is constrained to guarantee the eventual progress of the recovery line, and therefore processes may be forced to take additional checkpoints. The checkpoints that a process takes independently are called local checkpoints, while those that a process is forced to take are called forced checkpoints. Communication-induced checkpointing piggybacks protocol-related information on each application message. The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line. The forced checkpoint must be taken before the application may process the contents of the message, possibly incurring high latency and overhead. It is therefore desirable in these systems to reduce the number of forced checkpoints to the extent possible. In contrast with coordinated checkpointing, no special coordination messages are exchanged.

2.7 Model-based Checkpointing

Model-based checkpointing relies on preventing patterns of communications and checkpoints that could result in inconsistent states among the existing checkpoints. A model is set up to detect the possibility that such patterns could be forming within the system, according to some heuristic. A checkpoint is usually forced to prevent the undesirable patterns from occurring. Model-based checkpointing works with the restriction that no control (out-of band) messages are exchanged among the processes during normal operation. All information necessary to execute the protocol is piggybacked on top of application messages. The decision to force a checkpoint is done locally using the information available. Therefore, under this style of checkpointing it is possible that two processes detect the potential for inconsistent checkpoints and independently force local checkpoints to prevent the formation of undesirable patterns that may never actually materialize or that could be prevented by a single forced checkpoint. Thus, this style of checkpointing always errs on the conservative side by taking more forced checkpoints than is probably necessary, because without explicit coordination, no process has complete information about the global system state. 13 The literature contains several domino-effect-free checkpoint and communication models. The MRS model [50] avoids the domino effect by ensuring that within every checkpoint interval all message-receiving events precede all message-sending events. This model can be maintained by taking an additional checkpoint before every message-receiving event that is not separated from its previous message-sending event by a checkpoint [60]. Another way to prevent the domino effect is to avoid rollback propagation completely by taking a checkpoint immediately after every message-sending event [47]. Recent work has focused on ensuring that every checkpoint can belong to a consistent global checkpoint and therefore is not useless [5][24][25][41].

2.8 Index-based Communication Induced Checkpointing

Index-based communication – induced check pointing works by assigning monotonically increasing indexes to checkpoints, such that the checkpoints having the same index at different processes form a consistent state [14]. The indexes are piggybacked on application messages to help receivers decide when they should force a checkpoint. For instance, the protocol by Briatico et al forces a process to take a checkpoint upon receiving a message with a piggybacked index greater than the local index [14]. More sophisticated protocols piggyback more information on top of application messages to minimize the number of forced checkpoints [24].

3. System Model

Most of the algorithms in distributed mobile systems use the common system model in which the system is composed of a set of n nodes, and a network of communication links connecting the nodes. Some of the nodes may change their location with time. They will be referred to as mobile hosts or MH [1, 3]. The static nodes are connected to each other by a static network. An MH can be directly connected to at most one MSS at any given time and can communicate with other MHs and MSSs only through the MSS to which it is directly connected. The links in the static network support FIFO message communication. As long as an MH is connected to an MSS, the channel between them also ensures FIFO communication in both the directions. Message transmission through these links takes an unpredictable, but finite amount of time. During normal operation, no messages are lost or modified in transit. The system does not have any shared memory or a global clock. Hence, all communication and synchronization takes place through messages. A distributed application consists of processes that communicate asynchronously with each other. These processes run on different nodes of the mobile system. The processes exchange information with each other through messages. For the application to run successfully, all the nodes on which the modules of the application are running should function properly. Node failures in the system are assumed to be fail-stop in nature. Henceforth, the term node will be used for both MHs and MSSs, unless explicitly stated otherwise. The messages generated by the underlying distributed application will be referred to as the computation messages. Messages generated by the nodes to advance checkpoints, handle failures, and for recovery will be referred to as the system messages. Also, when a message of either type reaches a node, the node has the ability to peek at the message contents before actually processing it. Hence the reception/arrival of a message and its processing by the receiving node may not necessarily happen at the same time. They are two distinct events. The arrival of a message is recorded only on its processing.

4 Checkpointing algorithms for Distributed Mobile Systems

Chandy and Lamport [9] in 1985 were the first persons who present the algorithm for global snapshot in distributed systems. They give algorithm using FIFO channel.

4.1 Chandy-Lamport Algorithm [9]: they use a control message know as marker for the node which has recorded its state. After it recorded it state, it send marker to all of its' outgoing links. The role of marker is to act as delimiters for the messages in the channels so that the channel state recorded by the process at the receiving end of the channel.

Marker-Sending Rule for a Process p: For each channel c, incident on, and directed away from p:

p sends one marker along c after p records its state and before p sends further messages along c.

Marker-Receiving Rule for a Process q: On receiving a marker along a channel c:

```
if (q has not recorded its state) then
begin
q records its state;
q records the state c as the empty sequence
end
else q records the state of c as the sequence of messages received along c after q state
was recorded and before q received the marker along c.
```

The recorded local snapshots can be put together to create the global snapshot in several ways. One policy is to have each process send its local snapshot to the initiator of the algorithm. Another policy is to have each process send the information it records along all outgoing channels, and to have each process receiving such information for the first time propagate it along its outgoing channels. All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

4.2 Optimization of Chandy-Lamport Algorithm: Several solutions of the global state detection have been proposed. Many of them are based on optimization of Chandy-Lamport algorithms. One is given by Nigamanth and A.G. Sivilotti [2]. They optimize the algorithms and presented their lazy snapshots algorithm.

4.2.1 Lazy Algorithm [2]: The new algorithm works as follows. On receiving a marker from process p, process q "remembers" the reception of a marker from p. It sends markers on all outgoing channels as usual. However, q does not need to record its local snapshot as yet. It postpones the recording of the local snapshot to a later time. q is forced to take a local snapshot only if q receives a message from a process p, from which it has already received a marker. By delaying the recording of a local snapshot, the number of in-transit messages is decreased. Thus, a process can reduce the amount of channel state that it needs to record with the snapshot. The ability to postpone recording local state also has the advantage of giving process flexibility in scheduling this potentially expensive task. There is one technical problem with the postponement as described, however. Consider the case of a process r that does not communicate with the rest of the system. This process could just perform some local computation, never sending or receiving messages to the other processes. In such a case, all other processes in the system could take their local snapshots, but the global snapshot cannot be calculating until records its local state. In order to force the global state collector to terminate, a third event can be added: A marker has been received on every incoming channel. The local snapshot triggered by this event will record the state of every incoming channel as empty. The global state that this algorithm collects is indeed consistent. The algorithm can be seen as a generalization of the Chandy-Lamport algorithm. It reduces the space complexity of the recorded channel state and permits flexibility in scheduling the potentially expensive task of recording local state.

4.2.2 Spezialetti - Kearns Algorithm [3]: they proposed an optimization of the Chandy-Lamport algorithm to combine concurrently initiated snapshots. This way, if multiple processes initiate snapshot windows concurrently, the processes will only need to take one local snapshot and distribute the same local snapshot to the different initiators. This algorithm assumes bidirectional channels in the system. The message complexity of snapshot recording is O(e) irrespective of the number of concurrent initiations of the algorithm. The message complexity of assembling and disseminating the snapshot is $O(n^2)$ where r is the number of concurrent initiations.

4.2.3 Venkatesan's Incremental Snapshot method [4]: Venkatesan [4] proposed an incremental approach to collecting global snapshots. Using this solution, each process maintains the most recent snapshot taken. A new local snapshot would then just involve combining the local state changes since the last snapshot with the most recent snapshot. This algorithm, however, assumes the presence of only a single initiator process.

The incremental snapshot algorithm assumes bidirectional FIFO channels, the presence of a single initiator, a fixed spanning tree in the network, and four types of control messages: *initsnap*, *snap-completed*, *regular*, and *ack* .*initsnap* and *snap-completed* messages traverse spanning edges. r*egular* and *ack* messages which serve to record states of non-spanning edges are not sent on those edges on which no computation message has been sent since the previous snapshot. Venkatesan's algorithm achieves lower bound in message complexity.

4.2.4 Helary algorithm [7]: Another extension to the Chandy-Lamport algorithm was proposed by Helary [7] in 1989. In this algorithm, snapshot windows are marked by using message waves. Every process in the system is visited by a wave control message, and this triggers the recording of local state at the process. As soon as a wave terminates, the next wave is initiated. It uses message inhibition to avoid an inconsistency in a global snapshot. After a process i has sent a marker on the outgoing channel to process j, it does not send any messages on this channel until it is sure that j has recorded its local state. This algorithm has a message complexity of *O(e)* to record a snapshot.

4.2.5 Ten H. Lai and Tao H. Yang algorithm [6]: Lai and yang proposed a basic algorithm for non FIFO channel in 1987. Their algorithm piggybacks markers on messages, computes states of channel by the differences of message history, and needs no control messages. They fulfill the requirement of marker by coloring scheme on computation message. They purposed that every process is initially white and become red during the recording of snapshot. In their algorithm every white process takes its snapshot at time when it received a red message. This ensure that no message sent by a process after it become red. Thus an explicit marker message is not required in this algorithm and the marker is piggybacked on computation messages using a colouring scheme. Each process has to record the entire message history on each channel as part of its local snapshot, thus increase the requirement of the space. So they suggested that only current sent and received message are required to store because previous snapshot is available but still they relies upon that each process can take a snapshot spontaneously.

4.2.6 Letian He – Yongqiang Sun algorithm [8]: in 1997 they presented their snapshot algorithm called general repeated snapshot algorithm. They presented a repeated snapshot algorithm for non FIFO asynchronous distributed systems.

Repeated Snapshot Algorithm: they assumed that the processes in the system form a ring and a initial process is a process in the underlying computation. When a process P_i is to initiate a snapshot (i,sno), it sends a token marked with snapshot number(i,sno) to itself. When a process P_j receives a token (i,sno), if it has not gotten snapshot (i,sno), it records its local states , marks all following basic messages and passes token (i,sno) to successor of $P_{j.}$ Before a process receives a message marked with (i, sno) if it has not gotten snapshot (i, sno), it records its local states and receives the message, or else it just receives the message. When token (i,sno) return to P_i, P_i computes the global state. They used a counter to count the record message. Message sending is counted adding one and receiving is counted minus one. Thus the sum of all counters is the number of messages in channels. The algorithm has two parts one is server side and other is client side. Server side runs on initiator and client side run on all processes in the system.

To get one snapshot this algorithm needs n pieces of control messages to transmit local states and message counters. Each process needs extra space for store and maintained a message counter and an integer vector of snapshot number for each process.

4.2.7 Mattern Algorithm for distributed snapshots with global time approximation [10] in 1993: This algorithm does not require channels to be FIFO or messages to be acknowledged. Only a small amount of storage is needed. An important application of a snapshot algorithm is global virtual time determination for distributed simulation.

In this algorithm he assumed that a single process initiates the snapshot algorithm. The initiating process becomes red spontaneously and then starts a virtual broadcast scheme by directly or indirectly sending (red) *control messages* to all processes in order to ensure that eventually all processes become red. Virtual broadcast schemes can be implemented in various ways, for example by superimposing a control computation on the underlying basic computation which uses a ring, a spanning tree, or a flooding scheme. Note that a white process can receive a red basic message before receiving a control message.

Because processes do not know whether and when they will receive red basic messages, a white process must be able to take a local snapshot at the moment it receives a red basic message. This local snapshot must reflect the local state *before* the receipt of the message. In practice, this should not be a problem. If it is not possible to "peek" at the message contents before actually receiving it in order to determine its color, it might be possible to take a local snapshot just after receiving the message and before changing the local state. Otherwise a white process must save relevant parts of the local state before receiving a message in order to reproduce the state before the receipt event of a red message. To catch the messages Lai and Yang proposed that a process keeps a record of all messages sent and all messages received along its incident channel. This scheme requires a large amount of space. But in Mattern method the messages in transit are precisely the white messages which are received by red processes. Therefore, whenever a red process gets a white message it can send a copy of it to the snapshot initiator. (This message may be sent directly to the initiator or routed on a superimposed control topology). After the snapshot initiator has received the last copy of all in-transit messages (and the local snapshots of all processes) it knows the complete snapshot of the system.

Actually the Mattern algorithm is based on vector clocks. The initiator ticks its local clock and selects a future vector time s for the global snapshot. It then broadcasts s and freezes all activity until it receives an acknowledgement from every process. After all acknowledgements are received, the initiator increases its vector clock to s and broadcast a dummy message to all processes. Each process increases its clock to a value upon receiving the dummy message.

4.2.8 Michel Raynal algorithm [11]: in 1989 Michel Raynal presented his work using the prime number as a tool to design snapshot algorithm. He has shown that one of major drawback in designing snapshot algorithm lies in inability for one or several processes to catch instantaneously some part of the global state of the system. He has shown that in some cases prime number can be used to make distributed observation allowing making consistent decision. They purposed two approaches one is mutual exclusion and second is termination detection algorithms.

4.2.9 Minwen Ji algorithm [12]: in 2005 Minwen Ji from HP laboratories published his work for instant snapshots in a Federated array of Bricks. In a federated array of bricks (FAB), a snapshot may involve tens to thousands of independent controllers or processors, and may be taken at a high frequency, e.g., once every 30 seconds for atomic updates in remote mirroring. Therefore, he has given an efficient distributed snapshot algorithm that can make the snapshot operations transparent to applications in FAB.

They proposed such an algorithm, which avoids pausing or aborting write requests by the novel use of a tentative data structure during the two phases commit of a snapshot creation. The snapshot operations are serializable with data operations (i.e., reads and writes), hence ensure consistency of the snapshots. Read-only operations on snapshots are optimized in common cases, only requiring communications to a small subset of the bricks, in particular, a single replica set or three bricks in FAB. The algorithm has been prototyped in FAB and has been tested with trace based experiments.

This algorithm handles external messages as well as internal ones also. In algorithm old state leaves in the original location and stores the new state in a new location.

4.2.10 Arup Acharya and B.R. Badrinath Algorithm[1]: they presented a simple algorithm that relies on causal delivery of messages to record a distributed snapshot of systems with N processes. Then this requires N control messages. The snapshot algorithm is:

1. *Token transmission by the initiator process, Pinit :* Pinit multicasts a *token* message to every process in the system, including itself. At Pinit, the multicast is followed by delivery of the *token* message.

2. *Token delivery at a recipient process Pj, j* 2 *{1. . .N}:* On delivery of the *token* message, Pj executes the following actions.

a. Pj records its local snapshot, LSj, which consists of
• its local state, *local_statej*,
• current values of RECDj and SENTj.
i.e, LSj: = {*local_statej*, RECDj, SENTj}
b. Pj sends *reply*(LSj) to Pinit .

3. *Assembling a global snapshot at Pinit:* The initiator, Pinit, waits till the delivery of *reply* (LSj) message from every process Pj. The local state of each process has already been recorded and is available at Pinit. The channel states are computed by Pinit as a sequence of message ids, using the values of SENT and RECD arrays recorded by every process as part of its local snapshot. Thus, the global snapshot *GS*
is computed as :
a. ∀j 2 {1. . .N}, *local_state*j 2 GS.
Local_statej is available at Pinit with the delivery of *reply*(LSj) message from Pj.
b. ∀j 2 {1. . .N}, *state*(Cinit,j) := _
i.e., the channel Cinit,j does not contain any
in-transit messages. This claim is validated
in the next section *(Claim C3)*.
c. ∀i 2 {1. . .N},8 j 2 {1. . .N}, i 6= init,
state(Ci,j) := {(RECDj[i] + 1),. . ., SENTi[j]}
i.e., SENTi[j] represents the number of messages sent by Pi to Pj, before Pi recorded its local snapshot; RECDj[i] is the number of messages sent from Pi that were delivered at Pj before Pj recorded its local snapshot. Thus, the sequence of messages sent on the channel Ci,j, whose message-ids range from RECDj[i]+1 to SENTi[j], are considered to be in-transit on channel Ci,j in the global snapshot GS.

Note that the values of the SENT and RECD arrays used above are the recorded values, and not the current values.

This protocol relies on causal order of message delivery to record a consistent snapshot of systems of N processes with N messages. In this protocol, the state of a channel is computed by the initiator, as the sequence of *message-ids* of the in-transit messages. The initiator does not know directly the contents of those messages. To find the contents, it must rely either on sender having kept copies of the messages they sent, or on receivers collecting those messages as they arrive; the initiator then needs to send a second round of control messages to all the processes, incurring an additional overhead of *O*(N) messages, to collect the contents of the in-transit messages.

4.2.11 Alagar Venkatesan Algorithm[13]: In 1994 the Alagar-Vmkatesan algorithm, channel states are recorded **as** follows.

(i) When a process receives the token, it takes its snapshot, initializes the state of all channels to empty, and returns a Done message to the initiator. Now onwards, a process includes a message received on a channel in the channel state only if it is an old message.

(ii) After the initiator has received a Done message from all processes, it broadcasts a Terminate message.

(iii) A process stops the snapshot algorithm after receiving a Terminate message.

An interesting observation is that a process receives all the old messages in its incoming channels before it receives the Terminate message. This is ensured by the underlying causal message delivery property.

Causal ordering property ensures that no new message is delivered to a process prior to the ***token*** and only old messages are recorded in the channel states.

4.2.12 Franco Zambonelli [14] Domino free Snapshot algorithm: To permit one process to consistently restore its execution from its latest local checkpoint before the fault, one must grant that all its local checkpoints are useful can belong to at least one consistent global checkpoint. Otherwise, the execution of the process must be rolled back in the past until a useful local checkpoint is found from which to build a consistent global checkpoint. Rollback propagation, often called the domino effect because of its recursive nature, limits forward execution progresses in presence of faults.

Franco zambonelli algorithm deals with on-line algorithms that grant domino-free recovery by monitoring the application execution and by forcing additional local checkpoints in processes, when the arrival of one message is likely to make some local checkpoint useless. Several well known checkpoint algorithms are presented and integrated within a single theoretical framework. The effectiveness of the algorithms was evaluated in a heterogeneous set of message passing applications. The main result was that none of the algorithms shows itself capable of reasonably limiting the number of forced checkpoints, thus introducing a high overhead on applications.

4.2.13 Prakash - Singhal [18]: Ravi Prakash and Mukesh Singhal proposed Maximum Global Snapshot with concurrent initiators in 1994. They proposed snapshot algorithm to handle multiple concurrent snapshot initiations. They referred to the two approaches as the intersection approach and the union approach respectively. In the union approach the latest local snapshots at the nodes are selected. Therefore, the global snapshot obtained is more recent than that obtained from the intersection approach. They showed that unlike Spezialetti and Kearns algorithm, the propagation of snapshot, requests by various initiators is not suppressed. The effect is equivalent to letting multiple nodes collect global snapshot concurrently and independently, and then combining the snapshots to obtain a maximal global snapshot.

4.2.14 Ravi Prakash and Mukesh Singhal algorithm[15]: in 1996 Ravi prakash and Mukesh Singhal presented Low cost checkpointing and failure recovery in Mobile computing systems. They presented a synchronous snapshot collection algorithm for mobile systems that neither forces every node to take a local snapshot, nor blocks the underlying computation during snapshot collection. They also proposed a minimal rollback algorithm in which the computation at a node is rolled back only if it depends on operations that have been undone due to the failure of node. Both algorithms have low communication and storage overheads and meet the low energy consumptions and low bandwidth constraints of mobile computing systems.

4.2.15 Cao-Weijia Jia- Cheung algorithm [19]: in 1997 Cao-Jia-Cheung presented their work for an algorithm for coordinated checkpointing in distributed systems. In the algorithm message propagation is replaced by multi stage multicasting where only the initiator disseminates the checkpointing request and the final decision. For the first phase, the algorithm works in stages. In each stage, the initiator constructs a set of processes to which the checkpointing request will be sent next. In this way the initiator can eliminate unnecessary message propagation by merging the cohorts' sets and resolving redundancy in message.

4.2.16 Yang-Sun-Sattar-Yang algorithm [20]: in 1998 Zhonghua Yang, Chengheng Sun, Abdul Sattar and Yanyan Yang have presented the algorithm for consistent global state for distributed mobile computations. They presented two algorithms for finding consistent global states of a distributed mobile system. The first is Prepare and Cut algorithm and second is Cut-Along-Tree algorithm. In both algorithms three set of messages, Prepare, cut and Resume are sent from the initiator to all processes and back to the initiator and back to processes. The sending of application messages is disabling during taking snapshot. Both algorithms use very low message overhead to handle mobility issues and in disconnection operations.

4.2.17 Quaglia-Ciciani-Baldoni algorithm [21]: in 1998 QCB presented their work on analysis of several communication induced checkpointing protocol working in a mobile computing systems. They compared with varying both the mobility assumptions and disconnection rate of the mobile hosts. They simulated also heterogeneous environments to point out the performance of the protocols in a broad variety of scenarios. They produced the result showing that index based protocols perform better than the two phase one and well address the scalability issue of a mobile setting. They also shows that among the index based protocols, the QBC (Quaglia-Baldoni-Cicini Protocols) shows the best performance due to the reduction of the differences between sequence numbers in different mobile hosts, which is obtained without adding control information.

4.2.18 Cao-Singhal [22] Mutable Checkpoint algorithm: in 2001 Cao and Singhal presented 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. In this way, taking a mutable checkpoint avoids the overhead of transferring large amounts of data to the stable storage at MSSs over the wireless network. They presented techniques to minimize the number of mutable checkpoints. By simulation results they show that the overhead of taking mutable checkpoints is negligible.
Based on mutable checkpoints, non-blocking algorithm avoids the avalanche effect and forces only a minimum number of processes to take their checkpoints on the stable storage.

4.2.19 Yoshifumi - Manabe algorithm [23]: in 2001 Yoshifumi Manabe presented his work for Consistent Global Checkpoint algorithm for distributed Mobile System. He shown that a checkpoint algorithm in which the amount of information piggybacked on program messages does not depend on the number of mobile processes. The number of checkpoints is minimized under two assumptions one is consistent global checkpoint is taken for concurrent checkpoint initiations second is a checkpoint is initiated at each handoff by mobile processes. It was just optimal among the generalizations of Chandy and Lamport distributed snapshot algorithm.

4.2.20 Cao-Chen-Zhang-He algorithm for Hybrid Systems [26]: in 2004 Jiannong Cao, Yifeng Chen, Kang Zhang and Yanxiang He presented an algorithm which was developed for integrating independent and coordinated checkpointing for application running in a hybrid distributed system containing multiple heterogeneous systems. The algorithm has many advantages mainly its easy to implement, no change is required for subsystems with coordinated checkpointing schemes and low extra workload for the coordinated checkpointing subsystem.

4.2.21 Neogy-Sinha-Das CCUML algorithm [27]: in 2004 Sarmistha Neogy, Anupam Sinha, and Pradip K Das presented CCUML Coordinated Checkpointing with Unacknowledged Message Logging algorithm. The algorithm constructs consistent checkpoints in a distributive manner. The protocol eliminates the occurrences of both missing and orphan messages. Also each checkpoint taken by a process contributes to a consistent global snapshot and hence only the last global snapshot has to be retained.

4.2.22 Agbaria - Sanders algorithm [28]: in 2004 Adnan Agbaria and William H.Sanders presented their works for a new distributed snapshot for mobile computing systems, which often have limited bandwidth and long latencies, and where the mobile hosts may roam among the different cells within the system. In addition they also proved the live ness and safety. In order to minimize the overhead of protocol they tried not to minimize the communication through the wireless bandwidth. They also keep the major work of the protocol to be done by MSSs not by the MHs. On the other hand because the protocol does not take any additional checkpoints the recovery line is limited to the latest cut of checkpoint. The protocol does not use the CIC technique which complicates the recovery mechanism so there are no timeouts. This algorithm has not any type of forced checkpoints.

4.2.23 Vinit A. Ogale, Try Algorithm [29]: in 2004 the Vinit A. Ogale presented his algorithm called Try, try till you succeed: multiple checkpointing and rollback in distributed systems. In this he presented a multiple checkpointing and recovery protocol for fault tolerance in distributed systems. It assumed that the fault trigger occurs in rare circumstances and it is highly probable that the fault will not reoccur in another run. He has given an online distributed algorithm for slicing a distributed computation. This can be used for predicate detection also. The proposed scheme is practical and the overhead for fault tolerance is reasonably low.

4.2.24 Adnan Agbaria algorithm [30]: in 2006 adnan agbaria presented his worked for new distributed snapshot protocols which was just improvement of Lamport and Chandy algorithm in 1985. This algorithm has significant benefits in reducing the software and hardware overheads of distributed snapshots. It reduces the number of accesses to the secondary storage due to message logging. He also compared it with CL and SaS algorithm and show that it reduce access to the secondary storage by more than 95 %.

4.2.25 Garg -Vijay Garg – Sabharwal [32]: in 2006 Rahul Garg, Vijay K, Garg and Yogish Sabharwal proposed the scalable algorithms for global snapshots in distributed systems. They has given three algorithms first is Grid Base second was Tree based and third was Centralized algorithm for global snapshot. The grid based algorithm uses O(N) space but only root of N messages per processor. The tree based algorithm required only O(1)space and O(log N low w)messages per processor where w is the average number of messages in transit per processor. The centralized algorithm requires only O(1) space and O(log w) messages per processor. They also show that their algorithms have applications in checkpointing, detecting stable predicates and implementing synchronizers. They implemented and recorded the total latency, message sizes and counts, initial deficit and number of rounds for three algorithms.

4.2.26 Bidyut – Rahimi- Liu algorithm [31]: in 2006 Bidyut Gupta, Shahram Rahimi and Ziping Liu presented their work for mobile computing systems. In that work they presented a single phase non blocking coordinated checkpointing suitable for moble systems. This algorithm produces a consistent set of checkpoints without the overhead of temporary checkpoints.

4.2.27 Lalit - P. Kumar algorithm for mobile distributive systems [37]: in 2007 Lalit Kumar Awasthi and P. Kumar presented a new algorithm for synchronous checkpointing protocol for mobile distributed systems. In the algorithm they reduced the useless checkpoints and blocking using a probabilistic approach that computes an interacting set of processes on checkpoint initiation. A process checkpoint if the probability that it will get a checkpoint request in current initiation is high. A few processes may be blocked but they can continue their normal computation and may send messages. They also modified methodology to maintain exact dependencies. They show that their algorithm imposes low memory and computation overheads on MHs and low communication overheads on wireless channels. It avoids awakening of a MH if it not required to take its checkpoint. A MH can remain disconnected for an arbitrary period of time without affecting checkpointing activity.

4.2.28 Mandal –Mukhopadhyaya algorithm [33]: in 2007 Partha sarathi Mandal and Krishnendu Mukhopadhyaya presented the algorithm for checkpointing using Mobile agents in Distributed Systems. Mobile agents offer an attractive option for designing checkpointing schemes. When a process want to take a checkpoint, it just creates one mobile agent. Concurrent initiations by multiple processes are allowed in this algorithm. The mobile agents intelligently move from one process to an other and take checkpoints for host proceses without any useless checkpoints. An agent moves along a DFS tree rooted at the creator of the agent.

4.2.29 Qiangfeng Jiang and D. Manivannan algorithm [34]: in 2007 the Qiangfeng Jiang and D. Manivannan presented an optimistic checkpointing and selective message logging approach for consistent global checkpoint collection in distributed systems. In this work they presented a novel quasi-synchronous checkpointing algorithm that makes every checkpoint belong to a consistent global checkpoint. Under this algorithm every process takes tentative checkpoints and optimistically logs messages received after a tentative checkpoint is taken and before the tentative checkpoint is finalized. Since tentative checkpoint can be taken any time and sorted in local memory, tentative checkpoints taken can be flushed to stable storage anytime before that checkpoint is finalized.

4.2.30 Bidyut-Rahimi-Ziping Liu algorithm for ring Network [35]: in 2008 Bidyut Gupta, Shahram Rahimi and Ziping Liu presented non blocking checkpointing and recovery algorithms for bidirectional networks. The purposed algorithm allowed the process to take permanent checkpoints directly, without taking temporary checkpoints

S.No.	Algorithm	Complexity	Features	Channel	Approach
1	Chandy & Lamport [9], 1985	Message complexity is O(e) and O(d) is time where d is diameter of graph.	Basic algorithm for global snapshot algorithm	FIFO	**Centralized**
2	Nigamanth and A.G. Sivilotti [2]	Less than Chandy and Lamport	flexibility of postponed a local snapshot, improvement in Chandy & Lamport algorithm	FIFO	**Centralized**
3	Spezialetti-Kearns [3], 1986	Message Complexity of recording is O(e) and assembling and disseminating the snapshot is O(r n^2)	Two Phase algorithm, support concurrent initiators, advanced of Chandy & Lamport algorithm	Bidirectional	**Centralized**
4	Venkatesan's Incremental Snapshot [4], 1989	Message complexity is Ω(number of edges + number of process)	snapshot windows are marked by using message waves vector information with regular messages	Bidirectional FIFO	**Centralized**
5	Helary algorithm[7],1989	Message complexity is O(e)	snapshot windows are marked by using message waves , snapshot windows are marked by using message waves	Non FIFO	**Centralized**
6.	Ten H. Lai and Tao H. Yang algorithm [6], 1987	O(|c|), here c is the set of channels in the systems	Marker systems, use Markers piggybacked on messages	Non FIFO	**Distributive**
7.	Letian He – Yongqiang Sun algorithm [8], 1997	Message complexity of control messages is O(n)	Repeated snapshot algorithm, attached number of snapshot to messages, uses token passing.	Non FIFO	**Distributive**
8.	Mattern Algorithm [10], 1993	Total no of message is O(|c|)], response time is 2n, total message space is O(n^2)	No message history required	Non FIFO	**Distributive**
9.	Michel Raynal algorithm [11], 1989	Same as Chandy & Lamport algorithm	Use concepts of prime number and give consistent state always	FIFO	**Distributive**
10.	Minwen Ji algorithm [12], 2005	Less with compare to others because only global quorum of responses is required	Two Phase commit protocol, developed for FAB(federated array of bricks)	Non FIFO	**Distributive**
11.	Arup Acharya and B.R. Badrinath Algorithm [1], 1992	O(n)	Basic algorithm using causal order of message, channel message contents are not known	Causal order	**Centralized**
12.	Alagar Venkatesan Algorithm[13], 1993	Requires 3n messages, 3 time units	Small messages used in snapshot process	Causal order	**Distributive**

13.	Franco Zambonelli [14],1998	Intolerable overhead	Domino free snapshot algorithm, forcefully take additional check point in process	NON FIFO	**Distributive**
14.	Ravi Prakash & Mukesh Singhal [18], 1994	Message complexity is $O(m*n^2)$, where n is number of node and m is concurrently initiate snapshot collection	It can handle concurrent initiation of snapshot collection by multiple nodes.	FIFO	**Distributive**
15.	Ravi Prakash & Mukesh Singhal [15], 1996	Low cost checkpoint	Every node is not required to take local snapshot, introduced inter node dependencies	FIFO	**Distributive**
16.	Cao, Jia, Jia, Cheung [19], 1997	Message complexity is 3n(n-1), and time complexity is 3n.	Less time and space complexity, also used for synchronous rollback operations	FIFO	**Distributive**
17.	Yang, Sun, Sattar, and Yanyan Yang [20], 1998	Message O(3*n*d) for Prepare-and-Cut algorithm and O(3(n-1)) for Cut-Along- Tree algorithm	Give two algorithms first is Prepare and Cut and second is Cut Along Tree	Not required to be FIFO	**Distributive**
18.	Cao and Mukesh Singhal [22], 2001	Message overheads is around $2*N_{min} * C_{air} + \min(N_{min}*C_{air}, C_{broad})$. Blocking time is 0.	Introduced Mutable checkpoint, which is neither tentative or permanent checkpoint.	FIFO	**Distributive**
19.	Yoshifumi manabe [23], 2001	$O(n^2)$	Advancement in Lamport and Chandy algorithm, in this amount of information not depend on no of processor	FIFO	**Centralized**
20	Cao, Chen, Zhang and Yanxiang He [26], 2004	Introduce low extra overhead to hybrid system	Algorithm for hybrid distributed systems	-	**Distributed**
21.	Sarmistha, Anupam and Das [27], 2004	Total cost is $N*C_{air} + 2* C_{broad}$	Checkpointing with unacknowledged message logging, no useless checkpoint, nonblocking algorithm	FIFO	**Distributive**
22.	Vinit A Ogale [29], 2004	Extra Message required is O(mn) here m is maximum number of events at each process where a local checkpoint is stored.	Online algorithm for slicing a computation	FIFO	**Distributive**
23.	Adnan Agbaria and Sanders [28], 2004	$O(n^2)$	No timeout, no additional checkpoints, no forced checkpoints	Non FIFO	**Distributive**
24.	Adnan Agbaria, [30] 2006	$O(n^2)$ reduced number of access to secondary storage due to message logging.	Modified lamport and Chandy algorithm	FIFO	**Distributive**

25.	Bidyut, Rahimi and Liu[31], 2006	Total beta cost is N_{min} * C_{air}	Produce consistent set of checkpoint without overhead of temporary checkpoints, with no useless checkpoint, non blocking algorithm	Channel can loss messages	**Distributive**
26.	Rahul Garg, VijayGarg, and Yogish sagharwal [32], 2006	Grid base uses O(n) space and only O($n^{1.5}$)messages per processor. Tree base uses O(1) space and only O(Log n log w)messages per processor and centralized required only O(1) space and O(low w)messages per processor	Produced three algorithm tree base, grid base and centralized algorithm and simulate them	Gird, tree base	**Distributive**
27.	Mandal and Mukhopadhyaya [33], 2007	Time complexity is O(n), Control message size for k concurrent initiations is O(n/k)	Uses intelligent mobile agents in distributed system over network topology	FIFO	**Distributive**
28.	Qiangfeng and Manivannan [34], 2007	fast response time and reduce overheads of checkpoints	Every process can take their local tentative checkpoint and store in local memory	Non FIFO	**Distributive**
29.	Bidyut, Rahimi and Liu [35], 2008	Control message is n+1 and execution time is (n/2 +1)	Directly permanent checkpoint without any temporary checkpoints in ring network	FIFO	**Distributive**
30.	Subba Rao, Naidu[36], 2008	Minimize every type of overheads	Messages are logged only within a specified interval	FIFO	**distributive**
31.	Gao, Deng and Che[38], 2008	Control message is zero	Use time to indirectly coordinate the creation of consistent state	FIFO	**distributive**
32.	Ajay D Kshemkalyani[39], 2009	Response time is O(log n), message is O(n log n)	Useful in large scale systems	Non FIFO	**distributive**
33.	Ajay D Kshemkalyani, [40], 2010	In Simple tree messages are O(n) and response time is O(log n). in Hypercube requires O(n log n) messages and has O(log n) response time.	Useful in large distributive systems like supercomputers, MIMD, required less message and response time.	Non FIFO	**Distributive**

C_{air} is cost of sending a message form one process to another process.
C_{broad} : is cost of broadcasting a message to all processes.
N_{min} : is number of processes that need to take checkpoints
w: is the average number of in-transit messages when the snapshot is taken.
e stands for number of edges in the graph.
n stands for number of processes.
r is number of concurrent initiations.

Table 1: showing comparison of different snapshot algorithms for distributed systems

global snapshot algorithms for large scale distributed systems. He compared his algorithm with Garg[32] and and whenever a process is busy it takes a checkpoint after completing its current procedure. The algorithm was designed and simulate for Ring network.

4.2.31 Suba Raoand and Naidu algorithm [36]: in 2008 Ch. D.V. Subba Rao and M.N. Naidu presented their work for checkpointing algorithm combined with selective sender based message logging. This algorithm is free from problem of lost messages. This algorithm tolerates permanent faults in the presence of spare processors. In their absence it tolerates only transient failures. The term selective implies that messages are logged only within a specified interval known as active interval, thereby reducing message logging overhead. This algorithm minimizes different overheads like checkpointing overhead, message logging overhead, recovery overhead and blocking overhead.

4.2.32 Gao-Deng-Che algorithm [38]: in 2008 Yanping Gao, ChanghuiDeng and Yandong Che Presented their work for an indes based algorithm using time coordination in mobile computing. They use integration of time base and index based checkpointing algorithm. The proposed algorithm does not use any control message. It is more efficient because it takes lesser number of checkpoints and does not need to compute dependency relationship. In time based checkpointing protocols there is no need to send extra coordination messages. However they have to deal with the synchronization of timers. This type of algorithm is suitable for applications where processes have low message sending rate.

4.2.33 Ajay D Kshemkalyani algorithm [39]: in 2010 Ajay D. Kshemkalyani presented a fast and message efficientshow that new algorithm is more efficient. He presented two new algorithms Simple Tree and Hypercube that use fewer message and have lower response time and parallel communication times. In addition the hypercube algorithm is symmetrical and has greater potential for balanced workload and congestion freedom. This algorithm have direct applicable in large scale distributed systems such as peer to peer and MIMD supercomputers which are a fully connected topology of a large number of processors. This algorithm is also useful for determine checkpoint in large scale distributed mobile systems.

4.2.34 Ajay D. Kshemkalyani algorithm [40]: in 2010 Ajay D. Kshemkalyani has presented his work on large scale distributed systems and give two approaches, first are Simple Tree and second is Hypercube. He has shown that the response time and message complexity is minimum in these cases. Both algorithms are fast and required small numbers of message, this property make them highly scalable. The applications of this algorithm are in supercomputers and in MIMD processors.

5. Conclusion

A survey of the literate on checkpointing algorithms for distributed systems shows that a large number of papers have been published. A majority of these algorithms are based on the seminal article by chandy and lamport and have been obtained by relaxing many of the assumptions made by them. The table 1 gives a comparison of the salient features of various snapshot recording algorithms. Clearly, the higher the level of abstraction provided by a communication model, the simpler the snapshot algorithm. The requirement of global snapshots finds a large number of applications like: detection of stable properties, checkpointing, monitoring, debugging, analyses of distributed computation, discarding of obsolete information, etc. 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

[1] A. Acharya and B.R. Badrinath, ªCheckpointing Distributed Applications on Mobil Computers,º Proc. Third Int'l Conf. Parallel and Distributed Information Systems, Sept. 1994.

[2] Nigamanth Sridhar and Paolo A.G. Siviloti: Lazy Snapshots

[3] Spezialetti M and Keams P 1986 Efficient distributed snapshots *Proc.* 6th *Int. Conz* on *Distributed Computing Systems* pp 382-8

[4] Venkatesan *S* 1993 Message-optimal incremental snapshots *J. Comput. Sofnvare Engineering* 1 211-31

[5] B. Bhargava, S.R. Lian, and P.J. Leu, ªExperimental Evaluation of Concurrent Checkpointing and Rollback-Recovery Algorithms,º Proc. Int'l Conf. Data Eng., pp. 182-189, 1990.

[6] Lai T H and Yang T H 1987 On distributed snapshots *Informarion Processing Leu.* 25 153-8

[7] Helary I-M 1989 Observing global states of asynchronous distributed applications *Proc. 3rd Inr. Workhop on Disrribured A/gorirhnu, LNCS* 392 (Berlin: Springer) pp 124-34

[8] Letian He and Yongqiang Sun: A General Repeated Snapshot Algorithm IEEE, 1997

[9] K.M. Chandy and L. Lamport, ªDistributed Snapshots: Determining Global States of Distributed Systems,º ACM Trans. Computer Systems, Feb. 1985. Mattem F,1993 Efficient algorithms for distributed snapshots and global virtual lime approximation

[10] *J,Parallel Disrribured Computing* 18 423-34

[11] Michel Raynal : Prime Number as a tool to design distributed algorithms

[12] Minwen Ji: Instant Snapshots in a Fedeerated Arrary of Bricks, 2005

[13] Alagar S and Venkatesan *S* 1994 An optimal algorithm for distributed snapshots with causal message ordering *Infomation Processing Lert. 50* 3116

[14] Franco Zambonelli: On the Effectiveness of Distributed Checkpoint Algorithms for Domino Free Recovery, IEEE, Proceeding of HPDC-7,98, July 1998, at Chicago

[15] R. Prakash and M. Singhal, ªLow-Cost Checkpointing and Failure Recovery in Mobile Computing Systems,º IEEE Trans. Parallel and Distributed Systems, pp. 1035-1048, Oct. 1996.

[16] M. Satyanarayanan. Fundamental challenges in mobile computing In Fifteenth ACM Symposium on Principles of Distributed Computing, Philadelphia, PA, May, 1996

[17] Z.Yang and T.A. Marsland Global state and Time in Distributed Systmes. IEEE Computer Society Press 1994

[18] R. Prakash and M. Singhal, ªMaximal Global Snapshot with Concurrent Initiators,º Proc. Sixth IEEE Symp. Parallel and Distributed Processing, pp. 344-351, Oct. 1994.

[19] Jiannong Cao, Weijia Jia, and Xiaohua Jia, To-yat Cheung: Design and analysis of an efficient algorithm for coordinated checkpointing in distributed systems, IEEE, 1997

[20] Zhonghua Yang, Chengzheng Sun, Abdul Sattar, and Yanyan Yang: Consistent global states of distributed Mobile Computations, Proceedings of international conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, Usa, 1998

[21] F. Quaglia, B. Ciciani, R. Baldoni: Checkpointing Protocol in Distributed Systems with Mobile Hosts: a Performance Analysis,

[22] Guohong Cao, Mukesh Singhal "Mutable checkpoints: A new checkpointing approach for mobile computing systems" IEEE transactions on Parallel and distributed systems, 2001

[23] Yoshifumi Manabe: A Distributed Consistent Global Checkpoint Alagorithm for Distributed Mobile Systems, IEEE, 2001

[24] M. Banâtre, P. Heng, G. Muller and B. Rochard. "How to design reliable servers using fault-tolerant micro-kernel mechanisms." In *Proceedings of the USENIX Mach Symposium*, pp. 223—231, Nov. 1991.

[25] G. Barigazzi and L. Strigini. "Application-transparent setting of recovery points." In *Proceedings of the Thirteenth International Symposium on Fault-Tolerant Computing Systems*, *FTCS*-13, pp. 48—55, 1983

[26] Jiannog Cao, Yifeng Chen, Kang Zhang, Yanixing He: Checkpointing In Hybrid Distributed Systems, Proceedings of 7th international symposium of Parallel architetures, Algorithms and Network, IEEE, 2004

[27] Sarmistha Neogy, Anupam Siha, Pradip K Das: CCUML: A checkpointing protocol for distributed system proceses, IEEE, 2004

[28] Adnan Agbaria, William H. Sanders " Distributed snapshots for mobile computing systems" IEEE conference on Pervasive computing and communications, 2004.

[29] Vinit A. Ogale: Try,try till you succeed: Multiple checkpointing and rollback in distributed systems, 2004

[30] Adnan Agbaria: Improvements and Reconsideration of distributed snapshot protocols, 25th IEEE Symposium on Reliable Distributed Systems, IEEE Computer Society, 2006

[31] Bidyut Gupta, Shahram Rahimi, Ziping Liu: A new high performance checkpointing approach for mobile computing systems, International Journal of Computer science and network security, 2006

[32] Rahul Garg, Vijay K Garg, Yogish sabharwal "Scalable algorithms for global snapshots in distributed systems" ACM 2006

[33] Partha Sarathi Mandal and Krishenendu Mukhopadhyaya: Checkpointing using Mobile agents in distributed systems, Proceeding of international conference on Computing by IEEE Computer society, 2007

[34] Qiangfeng Jiang and D. Manivannan: An Optimistic Checkpointing and selective message logging approach for consistent global checkpoint collection in distributed systems, IEEE, 2007

[35] Bidyut Gupta, Shahram Rahimi, and Ziping Liu: Design of high performance distributed snapshot/recovery algorithms for ring network, Journal of Computing and information Technology-CIT, 2008

[36] Ch. D.V. Subba Rao and MM Naidu: A new, efficient corrdinated checkpointing protocol combined with selective sender based message logging, IEEE, 2008

[37] Lalit Kumar P. Kumar "A synchronous ckeckpointing protocol for mobile distributed systems: probabilistic approach" Int Journal of information and computer security 2007

[38] Yanping Gao, Changhui Deng, Yandong Che: an adaptive index based algorithm using time coordination in mobile computing, International symposiums on information processing, IEEE, 2008

[39] Ajay D Kshemkalyani: a symmetric O(n log n) message distributed snapshot algorithm for large scale systems, IEEE, 2010

[40] Ajay D Kshemkalyani " Fast and message efficient global snapshot algorithms for large scale distributed systems" IEEE 2010

[41] J. Cao and K. C. Wang. "An abstract model of rollback recovery control in distributed systems." *Operating Systems Review*, pp. 62—76, Oct. 1992.

[42] R.E. Strom and S.A. Yemini, ªOptimistic Recovery In Distributed Systems,º ACM Trans. Computer Systems, pp. 204-226, Aug. 1985.

[43] R. Koo and S. Toueg, ªCheckpointing and Rollback-Recovery for Distributed Systems,º IEEE Trans. Software Eng., pp. 23-31, Jan. 1987.

[44] M. Banâtre, P. Heng, G. Muller and B. Rochard. "How to design reliable servers using fault-tolerant micro-kernel mechanisms." In *Proceedings of the USENIX Mach Symposium*, pp. 223—231, Nov. 1991.

[45] G. Barigazzi and L. Strigini. "Application-transparent setting of recovery points." In *Proceedings of the Thirteenth International Symposium on Fault-Tolerant Computing Systems*, *FTCS*-13, pp. 48—55, 1983

[46] N. Attig and V. Sander. "Automatic checkpointing of NQS batch jobs on CRAY UNICOS systems." In *Proceedings of the Cray User Group Meeting*, 1993.

[47] L. Alvisi and K. Marzullo. "Message logging: Pessimistic, optimistic and causal." In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, May 1995.

[48] N. Attig and V. Sander. "Automatic checkpointing of NQS batch jobs on CRAY UNICOS systems." In *Proceedings of the Cray User Group Meeting*, 1993.

[49] M.S. Algudady and C.R. Das. "A cache-based checkpointing scheme for MIN-based multiprocessors." In *Proceedingsof the International Conference on Parallel Processing*, pp. 497—500, 1991.

[50] M. Choy, H. Leong, and M.H. Wong. "On distributed object checkpointing and recovery." In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Aug. 1995

[51] I. Akyildiz, J. Mcnair, J. Ho, H. Uzunalioglu, and W. Wang, ªMobility Management in Next-Generation Wireless Systems,º IEEE, vol. 87, no. 8. pp. 1347-1384, Aug. 1999.

[52] B. Bhargava and S. Lian, ªIndependent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems,º Proc. Seventh IEEE Symposium Reliable Distributed System, pp. 3-12, Oct. 1988.

[53] M.S. Algudady and C.R. Das. "A cache-based checkpointing scheme for MIN-based multiprocessors." In *Proceedings of the International Conference on Parallel Processing*, pp. 497—500, 1991.

[54] P. Ramanathan and K.G. Shin, ªUse of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System,º IEEE Trans. Software Eng., pp. 571-583, June 1993.

[55] P.Y. Leu and B. Bhargava, ªConcurrent Robust Checkpointing and Recovery in Distributed Systems,º Proc. Fourth IEEE Int'l. Conf. Data Eng., pp. 154-163, 1988.

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

[57] N. Vaidya, ªStaggered Consistent Checkpointing,º IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 7, pp.694-702, July 1999.

[58] Y. Wang and W.K. Fuchs, ªLazy Checkpoint Coordination for Bounding Rollback Propagation,º Proc. 12th Symp. Reliable Distributed Systems, pp. 78-85, Oct. 1993.

[59] Y. Deng and E.K. Park, ªCheckpointing and Rollback-Recovery Algorithms in Distributed Systems,º J. Systems and Software, pp. 59- 71, Apr. 1994.

[60] A. Duda. "The effects of checkpointing on program execution time." In *Information Processing Letters*, no. 16, pp. 221—229, 1983.

[61] B. Crow, I. Widjaja, J. Kim, and P. Sakai, ªIEEE 802 11 Wireless Local Area Networks,º IEEE Comm. Magazine, pp. 116-126, Sept.1997.

[62] G.H. Forman and J. Zahorjan, ªThe Challenges of Mobile Computing,º Computer, pp. 38-47, Apr. 1994