Volume: 07 Issue: 02 Pages: 2682-2689 (2015) ISSN: 0975-0290

A Survey of Various Fault Tolerance Checkpointing Algorithms in Distributed System

Sudha
Department of Computer Science, Amity University Haryana, India
Email: sudhayadav.91@gmail.com
Nisha
Department of Computer Science, Amity University Haryana, India
Email: yadavnisha021@gmail.com

ABSTRACT

A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved. Checkpoint is defined as a fault tolerant technique. It is a save state of a process during the failure-free execution, enabling it to restart from this checkpointed state upon a failure to reduce the amount of lost work instead of repeating the computation from beginning. The process of restoring form previous checkpointed state is known as rollback recovery. A checkpoint can be saved on either the stable storage or the volatile storage depending on the failure scenarios to be tolerated. Checkpointing is major challenge in mobile ad hoc network. The mobile ad hoc network architecture is one consisting of a set of self configure mobile hosts(MH) capable of communicating with each other without the assistance of base stations, some of processes running on mobile host. The main issues of this environment are insufficient power and limited storage capacity. This paper surveys the algorithms which have been reported in the literature for checkpointing in distributed systems as well as Mobile Distributed systems.

Keywords – **Checkpointing, Distributed systems, Fault tolerance, Mobile computing system, Rollback recovery.**

Date of Submission: Sep 04, 2015 Date of Acceptance: Oct 01, 2015

I. INTRODUCTION

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 [4]. In a traditional distributed system all hosts are stationary but recent techniques in portable computers with wireless communication interfaces and satellite services have made it possible for mobile users to execute distributed applications and to access information anywhere and at anytime [2].This new computing environment where some hosts are mobile computers connected via wireless networks and some hosts are stationary connected via a fixed network is called a distributed mobile computing environment. The infrastructure machines that communicate directly with the MHs are called Mobile Support Stations (MSSs). A geographical or logical coverage area under an MSS is called Cell. Distributed computing is being used extensively as they are cost-effective and scalable, and are able to meet the demands of high performance computing. 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.

With the increase in the number of components there is an increase in the failure probability. To provide fault tolerance it is essential to understand the nature of the faults that occur in these systems [2].

Types of faults occur in the system: Faults are mainly of two types: 1) Permanent 2) Transient. Permanent faults are caused by permanent damage to one or more components like hardware failure and transient faults are caused by changes in environmental conditions. Permanent faults can be rectified by repair or replacement of components. Transient faults remain for a short duration of time and are difficult to detect and deal with but not lead to a permanent damage. Recovery from permanent faults must include replacement of the damaged part and reconfiguration of the system but in case of Recovery from transient faults, it is comparatively simple as compared to the permanent faults because reconfiguration of the system is not needed [5], [26].

Fault tolerance can be achieved through some kind of redundancy. Redundancy can be temporal or spatial. In temporal redundancy, i.e., checkpoint-restart, an application is restarted from an earlier checkpoint or recovery point after a fault. This may result in the loss of some processing and applications may not be able to meet strict timing targets. In spatial redundancy, many copies of the application execute on different processors concurrently and strict timing constraints can be met. But the cost of providing fault tolerance using spatial redundancy is quite high and may require extra hardware. Checkpointing is primarily used to avoid losing all the useful processing done before a fault has occurred.

Checkpointing consists of intermittently saving the state of a program in a reliable storage medium. Upon detection of a fault, previous consistent state is restored. In case of a fault, checkpointing enables the execution of a program to be resumed from a previous consistent state rather than resuming the execution from the beginning [5].

During the designing of checkpoint protocols for distributed mobile systems following features must be take care into account otherwise the protocol will incur high overheads or it will not work correctly:

1. Designs of mobile computing need to be very concerned about bandwidth consumption.
2. Mobility is inherently vulnerable so instead to store local checkpoints on MH, store them to MSS to which the mobile host is connected. This would require each of the mobile hosts to take their checkpoints and transfer them to their MSSs.
3. Mobile hosts are often disconnected from the rest of the system or frequently disconnect by going into low energy mode. A disconnected mobile host can neither send nor receive messages, but can continue an application execution by using its local data and cashed shared data.
4. The mobility implies that a mobile host may change its location during distributed computation because of this they have to be searched and located before control messages associated with the checkpointing [2].

II. CHECKPOINT CLASSIFICATION

Processes in a distributed system communicate by sending and receiving messages. A process can record only its own state and messages it sends and receives. 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. To determine a global system state, a process Pi must enlist the cooperation of other processes that must record their own local states and send the recorded local states to Pi. All processes cannot record their local states at precisely the same instant unless they have access to a common clock. We assume that processes do not share clocks or memory. The problem is to devise algorithms by which processes record their own states and the states of communication channels so that the set of process and channel states recorded form a global system state [5].

Depending on the programmer's intervention in process of checkpointing, the classification can be:

- User-Triggered checkpointing
- Transparent Checkpointing

User triggered checkpointing [6] schemes require user interaction and are useful in reducing the stable storage requirement. These are generally employed where the user has the knowledge of the computation being performed and can decide the location of the checkpoints. The main problem is the identification of the checkpoint location by a user.

The transparent checkpointing techniques do not require user interaction and can be classified into following categories:
• Uncoordinated Checkpointing
• Coordinated Checkpointing
•Quasi-Synchronous or Communication induced Checkpointing
• Message Logging based Checkpointing

2.1 Uncoordinated or independent checkpointing [10], [16], [22], in this, processes do not coordinate their checkpointing activity and each process records its local checkpoint independently. It allows each process the maximum autonomy in deciding when to take checkpoint, i.e., each process may take a checkpoint when it is most convenient. It eliminates coordination overhead all together and forms a consistent global state on recovery after a fault. After a failure, a consistent global checkpoint is established by tracking the dependencies. There are several disadvantages also. First, it may require cascaded rollbacks that may lead to the initial state due to domino-effect. Second, it requires multiple checkpoints to be saved for each process and periodically invokes garbage collection algorithm to reclaim the checkpoints that are no longer needed. In this scheme, a process may take a useless checkpoint that will never be a part of global consistent state. Useless checkpoints incur overhead without advancing the recovery line. The main disadvantage of this approach is the domino-effect [Figure 1]. In this example, processes P1 and P2 have independently taken a sequence of checkpoints. The interleaving of messages and checkpoints leave no consistent set of checkpoints for P1 and P2, except the initial one at {C10, C20}. Consequently, after P1 fails, both P1 and P2 must roll back to the beginning of the computation. It should be noted that global state {C11, C21} is inconsistent due to orphan message m1. Similarly, global state {C12, C22} is inconsistent due to orphan message m4.

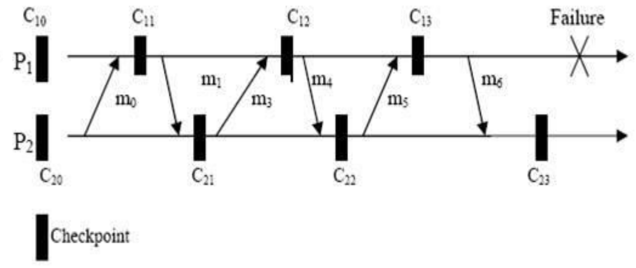

Fig. 1 Domino-effect

2.2 Coordinated Checkpointing [17], [20], [40], in coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows two-phase commit structure. In the first phase, processes take tentative checkpoints and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In case of a fault, processes rollback to last checkpointed state. Permanent checkpoint cannot be changed it means computation which have been completed till permanent checkpoint will not repeat again. Tentative checkpoint can be change. Its main disadvantage, however, is the large latency involved in committing output, since a global checkpoint is needed before messages can be sent to outside world.

First phase: **-** A coordinator takes a checkpoint and then broadcast a request message to all processes, asking them to take a checkpoint. When a process receives the message, it stops its executions, flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgement message back to the coordinator.

Second phase:-After the coordinator receives acknowledgements from all processes, it broadcasts a commit message that completes the two-phase checkpoint protocol. On receiving commit, a process converts its tentative checkpoint into permanent one and discards its old permanent checkpoint, if any. The process is then free to resume execution and exchange messages with other processes.

Coordinated checkpointing is of 2 types:-

1. Blocking: - A straightforward approach to coordinated checkpointing is to block communications while the checkpointing protocol executes. In this type, communication will be block between the processes during execution of checkpoint protocol because prevent a process from receiving application messages that could make the checkpoint inconsistent [40].

2. Non-blocking:-It will overcome the drawback of blocking coordinated checkpointing algorithm. No blocking required for processes during execution of checkpointing algorithm. 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 [10], [16].

Minimum process checkpoint: Normally, a coordinated checkpointing impose all process to participate in every checkpointing. 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, only those process need to take new checkpoints not all. In the first phase, the checkpoint initiator identifies all processes with which it has communicated since the last checkpoint and sends them a request. After 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 [40].

2.3 Quasi-Synchronous or Communication Induced Checkpointing [12], [35] this type of checkpointing avoids the domino-effect while allowing processes to take some of their checkpoints independently without requiring all checkpoints to be coordinated. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken independently, while forced checkpoints are taken to guarantee the eventual progress of the recovery line and to minimize useless checkpoints. Communication-Induced checkpointing algorithm 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. Unlike coordinated checkpointing, no special coordination messages are exchanged.

2.4 Message logging [8], [9], [15], [33], [39] It is used to provide fault tolerance in distributed systems in which all inter-process communication is through messages. Each message received by a process is saved in message log 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 state needs to be consistent with the states of the other processes [13]. This consistency obligation is usually expressed in terms of orphan processes, which are surviving processes whose states are inconsistent with the recovered states of crashed processes. Thus, message- logging protocols guarantee that upon recovery, no process is an orphan. This requirement can be enforced either by avoiding the creation of orphans during an execution, as pessimistic protocols do, or by taking appropriate actions during recovery to eliminate all orphans as optimistic protocols do. Bin Yao et al. describes a receiver based message logging protocol for mobile hosts, mobile support stations and home agents in a Mobile IP environment, which guarantees independent recovery. Checkpointing is utilized to limit log size and recovery latency.

III. CHECKPOINTING ALGORITHMS FOR DISTRIBUTED MOBILE SYSTEMS

3.1 Chandy and Lamport [CL] proposed an algorithm for distributed systems which construct a global snapshot of the system [8]. It is an all-process non-blocking coordinated checkpointing scheme for distributed systems. In their algorithm, system messages (markers) are sent along all channels in the network during checkpointing.

This leads to a message complexity of O (N^2). Moreover, it requires all processes to take checkpoints and the channel must be FIFO. It is observed that most of the checkpointing algorithms proposed for message passing system uses CL algorithm as the base. CL algorithm is based on the following assumptions [17].

- The distributed system consists of a finite number of processes and a finite number of channels.
- The processes communicate with each other by message passing through communication channels.
- The channels are fault-free and Communication delay is arbitrary but finite.
- The global state of the system includes the local states of the processors and the state of the communication channels.
- State of a channel refers to the set of messages sent along that channel
- Buffers are of infinite capacity.
- Termination of the algorithm is ensured by fault-free communication.

The steps are below:

(1) Save the local context in a stable storage.
(2) For i = 1 to all outgoing channels
do {send markers along channel i};
(3) Continue regular computation;
(4) For i=1 to all incoming channels
do {Save incoming messages in channel i until a marker I is received along that channel}.

3.2 Acharya et al. [6], [9] in 1994, he was the first to present an asynchronous snapshot collection algorithm for distributed applications on mobile computing systems. Due to these two reasons they consider synchronous checkpointing to be unsuitable for mobile systems: (i) In the Chandy-Lamport [8] kind of algorithm, an MH has to receive REQUESTs along every incoming link so a high cost is required for locating an MH (ii) non-availability of the local snapshot of a disconnected MH during synchronous checkpointing.

The two phase based protocol from Acharya Badrinath [6] actually adapts the Russel Protocol [16] to the context of mobile systems. Each MH (h_i) owns a Boolean variable ‘phasei’ which can assume only two values (SEND and RECV). Upon receiving a message, if the value of ‘$phase_i$’ is SEND then h_i takes a check point and $phase_i$ is set to RECV. Every time whenever h_i will send a message then ‘$phase_i$’ will set to SEND. Acharya and Badrinath proved that to keep of the consistent global checkpoint a local checkpoint belongs to, in this protocol, a vector of integers must be piggybacked on each application message, which takes into account the causal dependency established between local checkpoints.

3.3 Lai-Yang Coloring Scheme They were present a global snapshot algorithm for non-FIFO systems and it is based on two observations on the role of a marker in a FIFO system. The Lai-Yang algorithm fulfils this role of a marker in a non-FIFO system by using a coloring scheme on computation messages that works as follows [11]:

1. Initially, every process is white and then turns into red while taking a snapshot. The equivalent of the “marker sending rule” is executed when a process turns red.
2. Every message sent by a white (red) process is coloured white (red). Thus a white (red) message is a message that was sent before (after) the sender of that message recorded its local snapshot.
3. Every white process takes its snapshot at its convenience, but no later than the instant it receives a red message.

Thus, when a white process receives a red message, before processing the message it records its local snap-shot This ensures that no message sent by a process after recording its local snapshot is processed by the destination process before the destination process records its local snapshot. Thus, an explicit marker message is not required in this algorithm and the “marker” is piggybacked on computation messages using a colouring scheme.

3.4 Koo-Toueg’s Minimum process Blocking Scheme [40] as its name indicates they proposed a minimum process blocking checkpointing algorithm for distributed systems. This algorithm makes the following assumption about distributed system: processes communicate by exchanging messages through communication channels and channels are FIFO. Communication failure does not partition the network. The algorithm consists of two phases. 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 checkpoint 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, because it’s a blocking algorithm so no process can send any message after taking a checkpoint until the second phase terminates successfully, although receiving messages after the checkpoint is permissible.

3.5 Cao-Singhal Non-intrusive Checkpointing Algorithm [29] had proposed an efficient minimum-process and non-blocking algorithm and it significantly reduces the number of checkpoints. Their algorithm requires minimum number of processes to take tentative checkpoints and thus minimizes the workload on stable storage server. Their algorithm has three kinds of checkpoints: tentative, permanent and forced. Tentative and permanent checkpoints are saved on stable storage. Forced checkpoints do not need to be saved on stable storage. They can be saved on any where even in the main memory. When a process takes a tentative checkpoint; it forces all dependent processes to take checkpoints. However a process taking a forced checkpoint does not require its dependent processes to take checkpoint. Thus taking a forced checkpoint avoids the cost of transferring large amount of data to stable storage and accessing the stable storage device and thus it has much less overhead compared to taking a tentative checkpoint on stable storage. Also by taking forced checkpoints their algorithm avoids avalanche effects (in avalanche effects the processes in the system recursively ask others to take checkpoints) and significantly reduces number of checkpoints. A process takes a forced checkpoint only when it receives a computation message which has a checkpoint sequence number larger than the process expects. Their algorithm is efficient in the sense that it is non-blocking, requires minimum stable storage, minimizes number of tentative checkpoints and avoids avalanche effect.

3.6 Silva and Silva Algorithm [15], [17] they proposed all process coordinated checkpointing protocol for distributed systems. The non- intrusiveness during checkpointing is achieved by piggybacking monotonically increasing checkpoint number along with computational message. As in [11], logic is same here but this algorithm use checkpoint number instead colouring scheme. When a process receives a computation message with the higher checkpoint number, it takes its checkpoint before processing the message. When it actually gets the checkpoint request from the initiator, it ignores the same. If each process of the distributed programs is allowed to initiate checkpoint operation, the network may be flooded with control messages and processes might waste their time making unnecessary checkpoints. In order to avoid this Silva and Silva give the key to initiate checkpoint algorithm to one process. The checkpoint event is triggered periodically by a local timer mechanism. When this timer expires, the initiator process checkpoints the state of processes running in its machine and forces all the others to take checkpoint by sending a broadcast message. The interval between adjacent checkpoints is called the checkpoint interval.

3.7 In Venkatesan's algorithm [13], [14] a node sends out markers (corresponding to REQUESTs in the proposed algorithm) on all the outgoing edges along which computation messages have been sent since the last checkpoint. However, in order to efficiently collect a consistent snapshot, checkpointing REQUESTs need only be propagated from the receiver of messages to the sender, not the other way round as in [24]. Moreover, in the snapshot collection algorithm of Koo-Toueg, [40]1987 only direct dependencies are maintained.

3.8 Juang-Venkatesan asynchronous checkpointing Scheme [5], [14] they gave an algorithm that is based on asynchronous checkpointing. During the recovery, we need to find a consistent set of checkpoints to which the system can be restored. In this recovery algorithm each process keeps track of both the number of messages it has send to and received from other processes. Several iterations of rollback by processes are also involved in this recovery. This algorithm avoids the existence of Orphan messages. Whenever a process rollbacks, it is necessary for all other processes to find if any message send by the rolled back process has become an orphan message. Orphan messages are discovered, if the number of messages received by processor Pi from process Pj is greater than number of messages sent by process Pj to process Pi, according to the current state of processes, then one or more message at process Pj are orphan messages. Then process Pj must rollback to a state where number of messages received are equal to the number of messages sent by the process.

3.9 Xu and Netzer Zig-Zag Paths [13] they introduced the concept of Zigzag paths, a simplification of Lamport's happened-before relation [5] and shown that notation of Zigzag path captures exactly the conditions for a set of checkpoints to belong to the same consistent global snapshot. They showed that a set of checkpoints can belong to the same consistent global snapshot if and only if no zigzag path exists from a checkpoint to any other checkpoint. If there exist a Zigzag path between a set of checkpoints belong to the same global state that means that global state is not consistent. But, if a global snapshot is consistent, then none of its checkpoints happened before the other. If we have two checkpoints such that none of them happened before other, it is still not sufficient to ensure that they can belong together to the same consistent snapshot. This happens when a zigzag path exists between such checkpoints. A zigzag path is defined as a generalization of Lamport's happened before relation [4].
Definition: A zigzag path exists from a checkpoint Cx,i to a checkpoint Cy,j iff there exists messages m1,m2,……mn(n≥1) such that

1. m1 is sent by process p_x after Cx,i;
2. if mk ($1 \le k \le n$) is received by process p_z, then mk+1 is sent by p_z in the same or a later checkpoint interval (although mk+1 may be sent before or after mk is received);
3. mn is received by process p_y before Cy,j.

3.10 In 1996 Prakash- Singhal [14], [28] proposed that a good checkpointing protocol for mobile distributed systems should have low memory overheads on MHs, low overheads on wireless channels and should avoid awakening of an MH in doze mode operations. The disconnection of MHs should not lead to infinite wait state. The algorithm should force minimum number of processes to take their local checkpoints.

3.11 Adnan Agbaria and William H. Sanders [23] in 2003 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 liveness and safety.

3.12 Garg and Sabharwal [7] in 2006, they proposed and proved three algorithms first is Grid Base second was tree based and third was centralized algorithm. 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.

3.13 Lalit and P. Kumar [17] they presented a new algorithm in 2007 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 taking its checkpoint. A MH can remain disconnected for an arbitrary period of time without affecting checkpointing activity.

3.14 Ajay D Kshemkalyani [38] he presented a fast and message efficient global snapshot algorithms for large scale distributed systems in 2007. He compared his algorithm with Garg and show 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.

3.15 Alvisi et. Message Logging Schemes [4], [20], [21], [26] He developed a message logging protocol that is non-blocking and avoids creation of orphan states. This protocol only sends the application messages and their acknowledgements. This scheme may make application messages arbitrarily larger, but it is claimed that average amount of overhead is small. The major limitation of this scheme is that it can only with stand a sequence of process crash, process recovery pairs. If process P sends messages to process Q and both P and Q simultaneously crash, then orphan states may be created and Q may find itself trying to reconstruct a message for which there exists only a receive sequence number.

3.16 Kim-Park Algorithm [10] proposed a time-efficient protocol for checkpointing recovery which exploits the dependency relationship between processes in checkpointing and rollback coordination. Unlike other synchronized protocols the coordinator of the checkpointing does not always have to deliver its decision after it collects the status of the processes it depends on hence one phase of the coordination is practically removed. The checkpointing coordination time and the possibility of total abort of the checkpointing are substantially reduced. Reduction of the coordination roll back time is also achieved by sending the restart messages from the coordinator directly to the roll back processes and concurrent activities of the checkpointing and roll back are effectively handled by exploiting the process dependency relationship.

3.17 Hybrid Coordinated Checkpointing Algorithm (24) In minimum-process checkpointing, some processes, having low communication activity, may not be included in the minimum set for several checkpoint initiations and thus may not advance their recovery line for a long time. In the case of a recovery after a fault, this may lead to their rollback to far earlier checkpointed state and the loss of computation at such processes may be exceedingly high. In all-process checkpointing, recovery line is advanced for each process after every global checkpoint but the checkpointing overhead may be exceedingly high, especially in mobile environments due to frequent checkpoints. MHs utilize the stable storage at the MSSs to store checkpoints of the MHs. Thus, to balance the checkpointing overhead and the loss of computation on recovery, a hybrid checkpointing algorithm for mobile distributed systems is proposed, where an all-process checkpoint is taken after certain number of minimum-process checkpoints.

A strategy is proposed to optimize the size of the checkpoint sequence number (csn). In order to address different checkpointing intervals, he replaced integer csn with k-bit CI. Integer csn is monotonically increasing, each time a process takes its checkpoint, it increments its csn by 1. K-bit CI is used to serve the purpose of integer csn. The value of k can be fine-tuned. The minimum-process checkpointing algorithm is based on keeping track of direct dependencies of processes. Initiator process collects the direct dependency vectors of all processes, computes minimum set, and sends the checkpoint request along with the minimum set to all processes. In this way, blocking time has been significantly reduced as compared to [37].

During the period, when a process sends its dependency set to the initiator and receives the minimum set, may receive some messages, which may alter its dependency set, and may add new members to the already computed minimum set. In order to keep the computed minimum set intact and to avoid useless checkpoints, he proposed to block the processes for this period.

3.18 Wang and Fuchs lazy checkpoint coordination [44]: They proposed a coordinated checkpointing scheme in which they incorporated the technique of lazy checkpoint coordination into an uncoordinated checkpointing protocol for bounding rollback propagation.

Recovery line progression is made by performing communication induced checkpoint coordination only when predetermined consistency criterion is violated. The notation of laziness provides a tradeoff between extra checkpoints during normal execution and average rollback distance for recovery.

3.19 Helary's Concept of Message Waves [45] He proposed a snapshot algorithm that uses the concept of message waves. A wave is a flow of control messages such that every process in the system is visited exactly once by a control message and at least one process in the system can determine when this flow of control messages terminates. Wave sequences may be implemented by various traversal structures such as a ring. A process begins recording the local snapshot when it is visited by the wave control message.

3.20 Elnozahy and Zwaenepoel Algorithm [46] they proposed a message logging protocol which uses coordinated checkpointing with message logging. The combination of message logging and coordinated checkpointing offers several advantages, including improved failure free performance, bounded recovery time, simplified garbage collection and reduced complexity.

IV. CONCLUSION

A survey of the literate on checkpointing algorithms for mobile distributed systems shows that a large number of papers have been published. A majority of these algorithms are based on the concept by chandy and lamport and have been obtained by relaxing many of the assumptions made by them. 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.

Checkpointing does not require the processes to coordinate their checkpoints, but it suffers from potential domino effect, complicates recovery, and still requires coordination to perform output commit or garbage collection. Between these two ends are communication-induced checkpointing schemes that depend on the communication patterns of the applications to trigger checkpoints. These schemes do not suffer from the domino effect and do not require coordination. Message logging based checkpointing avoid creation of orphan message during an execution and preserve consistency.

REFERENCES

[1] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, *The Performance of Consistent Checkpointing* Proc. *11th* Symp. Reliable.

[2] *A Survey of Checkpointing Algorithms in Mobile Ad Hoc Network* By Ms. Pooja Sharma & Dr. Ajay Khunteta.

[3] A. Borg, J. Baumbach, and S. Glazer," *A Message System Supporting Fault Tolerance*", Proc. Symp. Operating System Principles, pp. 90-99, ACM SIG OPS, Oct. 1983.

[4] "*Distributed Operating System"* by Mukesh singhal.

[5] "*A Review of Fault Tolerant Checkpointing Protocols for Mobile Computing Systems"* Rachit Garg Singhania University Dept. of Computer Sc & Engg Pacheri Bari (Rajasthan), India Praveen Kumar Meerut Institute of Engg & Tech. Dept of Computer Sc. & Engg Meerut (INDIA).

[6] Avi Ziv and Jehoshua Bruck, " *Checkpointing in Parallel and Distributed Systems*" Book Chapter from Parallel and Distributed Computing Handbook edited by Albert Z. H. Zomaya, pp. 274-302, (Mc Graw Hill, 1996).

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

[8] K.M. Chandy and L. Lamport, *ªDistributed Snapshots: Determining Global States of Distributed Systems*,º ACM Trans. Computer

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

[10] J.L. Kim and T. Park, ª*An Efficient Protocol for Checkpointing Recovery in Distributed Systems*,º IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[11] T.H. Lai and T.H. Yang, ª*On Distributed Snapshots,º Information Processing Letters, pp. 153-158*, May 1987.

[12] Baldoni R., Hélary J-M., Mostefaoui A. and Raynal M., "*A Communication- Induced Checkpointing Protocol that Ensures Rollback-Dependency Trackability*," Proceedings of the International Symposium on Fault-Tolerant-Computing Systems, pp. 68-77, June 1997.

[13] R. Netzer and J. Xu, ª*Necessary and Sufficient Conditions for Consistent Global Snapshots*,º IEEE Trans. Parallel and Distributed.

[14] S. Venketasan, "*Message-Optimal Incremental Snapshots* , " Computer and Software Engineering, vol.1, no.3, pp. 211-231, 1993.

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

[16] Wood, W.G., "*A Decentralized Recovery Control Protocol*", 1981 IEEE Symposium on Fault Tolerant Computing, 1981.

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

[18] Bhargava B. and Lian S. R., "*Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach*," Proceedings of 17th IEEE Symposium on Reliable Distributed Systems, pp. 3-12, 1988.

[19] L. Alvisi and K. Marzullo," *Tradeoffs in implementing Optimal Message Logging Protocol*", Proc.

15th Symp. Principles of Distributed Computing, pp. 58-67, ACM, June, 1996.
[20] 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.
[21] Storm R., and Temini, S., "*Optimistic Recovery in Distributed Systems*", ACM Trans. Computer Systems, Aug, 1985, pp. 204-226.
[22] Adnan Agbaria, Wiilliam H Sanders," *Distributed Snapshots for Mobile Computing Systems*", IEEE Intl. Conf. PERCOM'04, pp. 1-10, 2004.
[23]S. Venketasan and T.Y. Juang, "*Efficient Algorithms for Optimistic Crash recovery*", Distributed Computing, vol. 8, no. 2, pp. 105-114, June 1994.
[24] "*A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems*" by Parveen kumar.
[25] .P. Sistla and J.L. Welch," *Efficient Distributed Recovery Using Message Logging*", Proc. 18th Symp. Principles of Distributed Computing", pp 223-238, Aug. 1989
[26] 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.
[27] L. Lamport, ª*Time, Clocks and Ordering of Events in Distributed Systems*,º Comm. of the ACM, July 1978. cao and singhal: mutable checkpoints: a new checkpointing approach for mobile computing systems 171.
[28] 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.
[29] 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.
[30] L. Alvisi,Hoppe, B., Marzullo, K., "*Nonblocking and Orphan-Free message Logging Protocol* " Proc. of 23rd Fault Tolerant Computing Symp., pp. 145-154, June 1993.
[31] L. Alvisi," *Understanding the Message Logging Paradigm for Masking Process Crashes*," Ph.D. Thesis, Cornell Univ., Dept. of Computer Science, Jan. 1996. available as Technical Report TR-96-1577.
[32] Elnozahy and Zwaenepoel W, " *Manetho: Transparent Roll-back Recovery with Low-overhead, Limited Rollback and Fast Output Commit*," IEEE Trans. Computers, vol. 41, no. 5, pp. 526-531, May 1992.
[33] D. Johnson, "*Distributed System Fault Tolerance Using Message Logging and Checkpointing*," Ph.D. Thesis, Rice Univ., Dec. 1989.
[34] Elnozahy and Zwaenepoel W, " *On the Use and Implementation of Message Logging*," 24th int'l Symp. Fault Tolerant Computing, pp. 298-307, IEEE Computer Society, June 1994.
[35]Hélary J. M., Mostefaoui A. and Raynal M., "*Communication-Induced Determination of Consistent Snapshots*," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.
[36] Richard C. Gass and Bidyut Gupta," An *Efficient Checkpointing Scheme for Mobile Computing Systems*", European Simulation Symposium, Oct 18-20, 2001, pp. 1-6.
[37] Higaki H. and Takizawa M., "*Checkpoint-recovery Protocol for Reliable Mobile Systems*," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.
[38] Ajay D Kshemkalyani "*Fast and message efficient global snapshot algorithms for large scale distributed systems*" IEEE 2010.
[39] Elnozahy and Zwaenepoel W, "*On the Use and Implementation of Message Logging*," 24th int'l Symp. Fault Tolerant Computing, pp. 298-307, IEEE Computer Society, June 1994.
[40] Koo R. and Toueg S., "*Checkpointing and Roll-Back Recovery for Distributed Systems*," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.
[41] D. Johnson, "Distributed *System Fault Tolerance Using Message Logging and Checkpointing*," Ph.D. Thesis, Rice Univ., Dec. 1989.
[42] Manivannan D. and Singhal M., "*Quasi-Synchronous Checkpointing: Models, Characterization, and Classification*," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 7, pp. 703-713, July 1999.
[43] Yoshinori Morita, Kengo Hiraga and Hiroaki Higaki," *Hybrid Checkpoint Protocol for Supporting Mobile-to-Mobile Communication*", Proc. of the International Conference on Information Networking, 2001.
[44] Wang Y. and Fuchs, W.K., "*Lazy Checkpoint Coordination for Bounding Rollback Propagation*," Proc. 12th Symp. Reliable Distributed Systems, pp. 78-85, Oct. 1993.
[45] Hélary J. M., Mostefaoui A. and Raynal M., "*Communication-Induced Determination of Consistent Snapshots*," Proceedings of the 28th International Symposium on Fault-Tolerant Computing, pp. 208-217, June 1998.
[46] Elnozahy and Zwaenepoel W, "On *the Use and Implementation of Message Logging*," 24th int'l Symp. Fault Tolerant Computing, pp. 298-307, IEEE Computer Society, June 1994.

Biographies

Sudha

I am Pursuing My M.Tech(Computer science) Form Amity University Gurgaon, Haryana. I have completed my B.Tech(Information Technology) degree from Maharishi Dayanand University, Rohtak.

Nisha

I am Pursuing My M.Tech(Computer science) Form Amity University Gurgaon, Haryana. I have completed my B.Tech(Computer science) degree from Maharishi Dayanand University, Rohtak.