Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

Review of Some Checkpointing Schemes for Distributed and Mobile Computing Environments

Mr Raman Kumar
Mewar University, Chittorgargh (Raj)
Email: rmn.kmr1@gmail.com
Dr Parveen Kumar
Amity University Gurgaon (Haryana)
Email: pk223475@gmail.com

----------**ABSTRACT**----------

Fault Tolerance Techniques facilitate systems to carry out tasks in the incidence of faults. A checkpoint is a local state of a process saved on stable storage. In a distributed system, since the processes in the system do not share memory; a global state of the system is defined as a combination of local states, one from each process. In case of a fault in distributed systems, checkpointing enables the execution of a program to be resumed from a previous consistent global state rather than resuming the execution from the commencement. In this way, the sum of constructive processing vanished because of the fault is appreciably reduced. In this paper, we talk about various issues related to the checkpointing for distributed systems and mobile computing environments. We also confer various types of checkpointing: coordinated checkpointing, asynchronous checkpointing, communication induced checkpointing and message logging based checkpointing. We also present a survey of some checkpointing algorithms for distributed systems.

Keywords**: Check pointing algorithms; parallel & distributed computing; rollback recovery; fault-tolerant systems**.

Date of Submission: December 24, 2014 Date of Acceptance: February 15, 2015

---

1. INTRODUCTION

1.1 Distribution Systems and Mobile Systems

A distributed system is a compilation of self-governing entities that work together to solve a problem that cannot be individually solved. A distributed system can be characterized as a collection of mostly autonomous processors communicating over a communication network [24].

In a distributed system, there exists no system wide common clock (global clock). In other words, the notion of global time does not exist. Suppose a global (common) clock is available for all the processes in the system. In this case, two different processes can observe a global clock value at different instants due to unpredictable message transmission delays [24].

Due to the absence of global time and shared memory, it is difficult to reason about the temporal order of events in a distributed system. Hence, algorithms for a distributed system are more complicated to design and debug compared to algorithms for centralized systems. In addition, the nonexistence of a global clock makes it harder to accumulate up to- date information on the state of the whole system [24].

In mobile distributed computing system, some processes are operating on mobile hosts (MHs). An MH is a computer that may retain its connectivity with the rest of the distributed system through a wireless network while on move or it may disconnect. It requires integration of portable computers within existing data network. An MH can connect to the network from different locations at different times. The infrastructure machines that communicate directly with the MHs are called Mobile Support Stations (MSSs). A cell is a logical or geographical coverage area under an MSS. All MHs that have identified themselves with a particular MSS are considered to be local to the MSS. An MH can straightforwardly converse with an MSS (and vice-versa) only if the MH is physically positioned within the cell serviced by the MSS. At any given moment of time, an MH may logically belong to only one cell; its current cell defines the MH's location [25].

To throw a message from an MH h_1 to another MH h_2, h_1 first sends the communication to its local MSS over the wireless network. This MSS then forwards the communication to the local MSS of h_2 which forwards it to h_2 over its local wireless network [25].

Many Algorithms for MHs may make use of a handoff modus operandi: when an MH switches cell, MSS of the two cells carry out the handoff procedure. An MSS may preserve algorithm specific data structures on the behalf of a local MH. When an MH moves into a new cell, data structures from the previous MSS are transferred to the new MSS. For this to be realized, it is compulsory that the MH either inform the previous MSS of the ID of this new MSS or vice-versa. It should be like that an MH supply the ID of its previous MSS after inflowing the new cell with the *join ()* message [25].

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

Disconnection is handled in a parallel way to an MH switching cells. Disconnection is unlike from failure. Disconnections are optional or volunteer by nature, so an MH informs the system prior to its happening and executes an application-specific disconnection procedure if necessary. Disconnection can be deliberate or uncontrolled. The term "disconnection" implies a voluntary disconnection. An unexpected or unintentional disconnection is considered to be a failure [25].

1.2 Software Based Fault Tolerance

Fault tolerance can be achieved through some kind of redundancy. Redundancy can be temporal or spatial. In spatial redundancy or hardware-based fault tolerance, many copies of the application execute on diverse processors concurrently and stringent timing constraints can be met. But the outlay of providing fault tolerance using spatial redundancy is fairly high and may require additional hardware. In temporal redundancy or software-based fault tolerance, an application is restarted from a former checkpoint or recovery point after a fault. This may result in the thrashing of some processing and applications may not be able to meet stringent timing targets. Checkpoint-Restart or Backward error recovery is quite inexpensive and does not require extra hardware in general. Besides providing fault tolerance, checkpointing can be used for process migration, debugging distributed applications; job swapping, postmortem analysis and stable property detection [24].There are two software based fault tolerance approaches for error recovery:

- Forward Error Recovery
- Backward Error Recovery

In **forward error recovery** techniques, the character of errors and damage caused by faults must be wholly and precisely assessed and so it becomes possible to eliminate those errors in the process state and enable the process to move onward [27]. In distributed system, precise estimation of all the faults may not be possible. In backward error recovery techniques, the nature of faults need not be predicted and in case of error, the process state is restored to preceding error-free state. It is independent of the nature of faults. Thus, backward error recovery is more common recovery mechanism [24].

1.3 Definitions and Notations

Checkpoint: Checkpoint is defined as a selected place in a program at which normal process is broken up specifically to conserve the status information necessary to allow recommencement of processing at a later time. A checkpoint is a local state of a process saved on stable storage. By periodically invoking the checkpointing process, one can save the status of a program at regular intervals [24].

Rollback Recovery: If there is a failure one may restart computation from the last checkpoints, thereby, avoiding repeating computation from the commencement. The process of resuming computation by rolling back to a saved state is called rollback recovery [24, 27].

Global State: In a distributed system, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. The state of channels corresponding to a global state is the set of communication sent but not yet received [24, 27].

Orphan Message: A message whose receive event is recorded, but its send event is lost in the recorded global state.

Consistent Global State: A global state is said to be "consistent" if it contains no orphan message. To recover from a failure, the system restarts its execution from a previous consistent global state saved on the stable storage during fault-free execution. In distributed systems, checkpointing can be independent, coordinated or quasi-synchronous. Message Logging is also used for fault tolerance in distributed systems [24, 27].

Asynchronous Checkpointing: Under the asynchronous approach, checkpoints at each process are taken independently without any synchronization among the processes. Because of absence of synchronization, there is no guarantee that a set of local checkpoints taken will be a consistent set of checkpoints. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [27].

Coordinated Checkpointing: 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 [27]. In the first phase, processes take tentative checkpoints and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In the case of a fault, processes rollback to the last check pointed state.

Communication-induced Checkpointing: It avoids the domino-effect without requiring all checkpoints to be coordinated [24]. 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. As opposed to coordinated checkpointing, these protocols do no exchange any special coordination messages to determine when forced checkpoints should be taken. But, they piggyback protocol specific information [generally checkpoint sequence numbers] on each application message; the receiver then uses this information to decide if it should take a forced checkpoint.

Deterministic Systems: If two processes start in the same state, and both receive the identical sequence of inputs, they will produce the identical sequence outputs and will finish in the same state. The state of a process is thus completely determined by its starting state and by sequence of messages it has received [23, [24], [25].

Checkpoint Interval (CI): The i^{th} CI of a process denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint.

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

Direct Dependency among Processes: P_j is directly dependent upon P_k only if there exists m such that P_j receives m from P_k in the current CI and P_k has not taken its permanent checkpoint after sending m.

Minimum Set: A process P_i is in the minimum set only if checkpoint initiator process is transitively dependent upon it.

Minimum-process Coordinated Checkpointing Algorithms : In these algorithms**,** only a subset of interacting processes (called minimum set) are required to take checkpoints in an initiation.

Anti-Message: David R. Jefferson [29] introduced the concept of anti-message. Anti-message is precisely like an original message in format and content except in one field, its sign. Two messages that are identical except for opposite signs are called anti-messages of one another. All messages sent explicitly by user programs have a positive (+) sign; and their anti-messages have a negative sign (-). Whenever a message and its anti-message occur in the same queue, they immediately eradicate one another. Thus the result of enqueueing a message may be to shorten the queue by one message rather than lengthen it by one.

2. LITERATURE SURVEY

2.1 Chandy and Lamport Algorithm [1]

They designed a global snapshot algorithm for distributed systems. It is observed that every checkpointing algorithm proposed for message passing system uses Chandy and Lamport's algorithm as the foundation. The algorithms proposed in literature for message passing systems may be derived by relaxing various assumptions made by the demand modifying the way each step is carried out.

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.

Each step of CL algorithm can be modified to accommodate some improvements in basic global snapshot algorithm.

In Step 1 node saves its context in stable storage .The overhead associated with step one is context saving overhead**.** The objective of saving context in stable storage is to ensure its availability after a node failure .The overhead of context saving is proportional to the size of context and the time taken to access the stable storage. Context saving overhead can be reduced by (a) minimizing the context size and (b) overlapping context saving with computation.

In Step 2 markers are sent along all outgoing channels .The purpose of a marker is

(1) To inform the receiving node that a new checkpoint has to be taken;

(2) To separate the messages of the previous and current checkpoint interval.

2.2 Wang and Fuchs Lazy Algorithm [17]

The lazy new algorithm work as on receiving a marker from process p , process q, "remembers" the marker (marks the channel dirty) of marker from p. It sends markers on all outgoing channels as usual. However q does not need along all outgoing channels postpones the recording of its local state . Local state recording can be postponed to a later time .q is forced to take a local snapshot only if q receives a message from a process p. a marker from which it has already received. By delaying the recording of a local snapshot the number of in- transit message 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 deferment as described, however. Consider the case of a process r that does not converse with the rest of the system. This process could just execute 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 calculated until r 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.

2.3 Venkatesan's Incremental method [19]:

Venkatesan proposed an incremental approach to collecting global snapshots. Using this solution each approach maintains the most recent snapshots taken. A new local snapshot would then just involve combining the local state change since the last snapshot with the most recent snapshot. This algorithm assumes the presence of only a single initiator process.

2.4 Koo-Toueg Alogritham [5]:

Koo and Toueg have shown that if the nodes capture their local snapshots in an uncoordinated manner, it may not be possible to assemble a consistent global state from such snapshots. The rollback may lead to domino effect. In Synchronous snapshot collection algorithm a node initiates snapshot collection by taking its local snapshot then it sends request message to another nodes to take their snapshot. If the nodes maintain information about casual dependencies a minimal number of nodes have to take their local snapshots in reaction to such requirements. Koo and Toueg had presented such an algorithm which involves suspending the underlying computation during snapshot compilation .The nodes resume the underlying when the snapshot collection terminates. Koo and Toueg have planned methods to handle concurrent initiations of snapshot collection. They handle concurrent snapshot collection in following way. Once a node takes a local snapshot, it is unwilling to take a snapshot in response to another initiator. The node sends a negative response to all subsequent requests until the snapshot request is

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

made permanent or until the snapshot collection is aborted. Their algorithm makes the following assumption about distributed system: processes communicate by exchanging messages through communication channels. Communication channels are FIFO. Communication failure does not partition the network. The Checkpoint algorithm takes two kinds of checkpoints on stable storage: Permanent and tentative. A permanent checkpoint is a local checkpoint at a process and is a part of a consistent global checkpoint .A tentative checkpoint is a temporary checkpoint that is made permanent checkpoint on successful termination of checkpointing algorithm .In case of a failure processes rollback only to their permanent checkpoints for recovery .The algorithm assumes that no process fails during the execution of algorithm .The algorithm consists of two phases

First Phase

An initiating process P_i captures a tentative checkpoint and requests all other processes to capture tentative checkpoints. Each process informs P_i whether it succeeded in taking a tentative checkpoint. A process says "no" to a request if it fails to take tentative checkpoint which would be due to numerous reasons depending on underlying function. If P_i learns that all the processes have effectively taken tentative checkpoints P_i decides that all tentative checkpoints should be made enduring; otherwise P_i decides that all the tentative checkpoints should be superfluous.

Second Phase

P_i informs all the processes that their verdict had been reach at the end of first phase. A process on receiving the message from P_i will act consequently. Therefore, either all or none of processes move ahead the checkpoint by taking permanent checkpoints. The algorithm requires that after a process has taken a tentative checkpoint it cannot send communications related to underlying computation until it informed of P_i's verdict.

2.5 Silva L, Silva J Alogrithm [16]:

They designed an all process synchronous checkpointing protocol for distributed systems. The non- intrusiveness during checkpointing is attained by piggybacking ever-increasing checkpoint numbers along with computational messages. When a process receives a computational message with the high checkpoint number, it captures its checkpoint before processing the message. When it truly gets the checkpoint request from the initiator, it ignores the same. If each process of the distributed program is permitted to initiate the checkpoint operation, the network may be swamped with control messages and process might misuse their time capturing unnecessary checkpoints. In order to avoid this, Silva and Silva gave 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 checkpoint the state of process running in the machine and force all the others to take checkpoint by sending a broadcast message. The interval between adjacent checkpoints is called checkpoint interval.

2.6 J.L.Kim and T.Park Algorithm [11] :

Kim-Park Algorithm proposed a protocol for checkpointing recovery which exploits the dependency relationship between processes to achieve time-efficiency in checkpointing and rollback coordination. Unlike other synchronized protocols. In which the checkpointing coordinator collects the status information of the processes that it depends on and delivers its decision, the process in their protocol takes a checkpoint when it knows that all processes on which it computationally depends took their checkpoints. In this way, 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 process. and concurrent activities of the checkpointing and roll back are effectively handled exploiting the process dependency relationship.

2.7 Ravi Prakash and Mukesh Singhal Algorithm [7]:

They had described a Synchronous Snapshot compilation algorithm for Mobile Systems that neither forces every node to take a local snapshot nor blocks the essential computation during snapshot collection. If a node initiates snapshot collection, local snapshot of only those nodes that have directly or transitively affected the initiator, take their checkpoints. This paper presents that the global snapshot collection terminates within a finite time of its request and collected global snapshot is consistent. This paper presents a minimal rollback/recovery algorithm in which the computation at a node is rolled back only if depends on operations that have been undone due to failure of node(s).Both the algorithms have low communication and storage overheads and meet the low energy consumption and low bandwidth constraints of mobile computing systems. The synchronous snapshot collection algorithm that accounts for the mobility of the nodes .The algorithm forces a minimal set of nodes to take their snapshots and underlying computation is not suspended during snapshot collection .An interesting aspect of the algorithm is that it has lazy phase that enables nodes to take local snapshots in quasi-asynchronous fashion, after the coordinated snapshot collection phase is over. This further reduces the amount of computation that is rollback during recovery from node failure. The lazy phase advances the checkpoint slowly rather than in a burst. This avoids disagreement for the low bandwidth channels. This algorithm also considers the changing topology of network due to mobility of nodes. Here the Recovery algorithm is a compromise between two diverse recovery strategies – fast recovery with high

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

communication and storage overhead and slow recovery with very little communication overhead.

2.8 Coa and Singhal Minimum-process Blocking Scheme [6]

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

2.9 Cao-Singhal Non-intrusive Checkpointing Algorithm [2]:

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

Such algorithms rely on the two-phase commit protocol and save two kinds of checkpoints on the stable storage: tentative and permanent.

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

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

2.10 P. Kumar and L. Kumar algorithm [3] :

In Cao and Singhal algorithm number of useless checkpoint may exceeding high in some situation [2]. P. Kumar and L. Kumar proposed a new for Synchronous check pointing protocol for mobile distributed system [3]. They are able to maintain exact dependencies among processes and make an approximate set of interacting processes at the beginning. In this way the time to collect coordinated checkpoint is reduced. The number of useless check pointing and blocking processes is also reduced. 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 message..

Suppose, during the execution of the check pointing algorithm, *Pi* takes its checkpoint and sends *m* to *Pj*. *Pj* receives *m* such that it has not taken its checkpoint for the current initiation and it does not know whether it will get the checkpoint request. If *Pj* takes its checkpoint after processing *m*, *m* will become orphan. In order to avoid such orphan messages, they propose the following technique. If *Pj* has sent at least one message to a process, say *Pk* and *Pk* is in the tentative minimum set, there is a good probability that *Pj* will get the checkpoint request. Therefore, *Pj* takes its induced checkpoint before processing *m*. An induced checkpoint is similar to the mutable checkpoint [14]. In this case, most probably, *Pj* will get the checkpoint request and its induced checkpoint will be converted into permanent one. There is a less probability that *Pj* will not get the checkpoint request and its induced checkpoint will be discarded. Alternatively, if there is not a good probability that *Pj* will get the checkpoint request, *Pj* buffers *m* till it takes its checkpoint or receives the commit message. They have tried to minimize the number of useless checkpoints and blocking of the

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

process by using the probabilistic approach and buffering selective messages at the receiver end. Exact dependencies among processes are maintained. It abolishes the useless checkpoint requests and reduces the number of replica checkpoint requests as compared to [14].

2.11 Kumar and Khunteta Minimum-Process Coordinated Check pointing Protocol For Mobile Distributed System [21]

They designed a minimum process algorithm for mobile distributed where no useless checkpoint are taken and an effort has been made to optimize the blocking of processes. In this algorithm they proposed that during the checkpoint period, selective message are buffered at the receiver end. During its blocking period, a process allowed to perform its normal procedure. So with the help of this method blocking of process is minimum. They captured the transitive dependencies during the normal execution by piggybacking dependency vector onto computational message. So in this way they try to reduce the check pointing time by avoiding the formation of checkpoint tree.

2.12 Kumar and Garg Algorithm[18] : Minimum-process coordinated check pointing is a suitable approach to introduce fault tolerance in mobile distributed systems transparently. It may require blocking of processes, extra synchronization messages or taking some useless checkpoints. Checkpointing overhead may be exceedingly high in all-process checkpointing. To optimize both matrices, the check pointing overhead and the loss of computation on recovery, Kumar and Garg proposed a hybrid checkpointing algorithm, where an all process checkpoint is enforced after executing minimum-process algorithm for a fixed number of time. In the first phase, the MHs in the minimum set are required to take soft checkpoint only. Soft Checkpoint is stored on the disk of the MH and is similar to mutable checkpoint. In the minimum process algorithm, a process takes its forced checkpoint only if it is having a good probability of getting the checkpoint request; otherwise, it buffers the received messages.

In hybrid checkpointing algorithm all process coordinated checkpoint is taken after the execution of minimum-process coordinated check pointing algorithm for a fixed number of times. The number of useless checkpoints and blocking of processes are reduced in minimum-process check pointing. They proposed probabilistic approach to reduce the number of useless checkpoints and blocking of process. Thus, the proposed protocol is simultaneously able to reduce the useless checkpoints and blocking of processes at very less cost of maintaining and collecting dependencies and piggybacking checkpoint sequence numbers onto normal messages. Concurrent initiations of the proposed protocol do not cause its concurrent executions. They try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in Coordination with others.

2.13 Ch. D. V. Subba Rao and M.M. Naidu [10]:

Check pointing and message logging are the popular and general-purpose tools for providing fault-tolerance in distributed systems. The most of the Coordinated checkpointing algorithms available in the literature have not addressed about treatment of the lost messages and these algorithms suffer from high output commit latency. To overcome the above limitations, the authors propose a new coordinated checkpointing protocol combined with selective sender-based message logging. The protocol is free from the problem of lost messages. The term 'selective' implies that messages are logged only within a specified interval known as active interval, thereby reducing message logging overhead. All processes take checkpoints at the end of their respective active intervals forming a consistent global state. Outside the active interval there is no checkpointing of process state. This protocol minimizes different overheads i.e. checkpointing overhead, message logging overhead, recovery overhead and blocking overhead. Unlike blocking coordinated checkpointing, the disk contentions are less in the proposed protocol. In this protocol there exists Pinitiator, which coordinates with all the processes to take a consistent global checkpoint. Pinitiator is responsible for invoking the checkpoint operation periodically. It sends control messages, prepare checkpoint and take checkpoint messages to all other processes.

2.14 Ajay D. Kshemkalyani [14]:

He proposed a fast and Message-Efficient Global Snapshot Algorithms for Large-Scale Distributed Systems. He presented two algorithms: simple tree, and hypercube, that requires very less number of messages. In addition, the hypercube algorithm is symmetrical and has greater potential for balanced workload and congestion-freedom. This algorithm finds the direct application in large scale distributation system such as peer to peer system and MIMD super computer which are fully connected topology of a large no of processors. All the algorithms assume non-FIFO channels.

CONCLUSION:

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

REFERENCES

[1]. Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Systems," ACM Transaction on Computing Systems, vol. 3, No. 1, pp. 63-75, February 1985.

Volume: 06 Issue: 06 Pages: 2542-2548 (2015) ISSN: 0975-0290

[2]. G. Cao and M. Singhal,"Mutable Checkpoints:A New Checkpointing Approach for Mobile Computing Systems", IEEE Transactions On Parallel And D istributed Systems,Vol.12,No.2,February 2001,pp 157-172.
[3]. Lalit Kumar Awasthi, Kumar p. 2007 A Synchoronous Checkpointing Protocol For Mobile Distributed Systems. Probabilistic Approach. Int J. Information and Computer Security, Vol.1, No.3 .pp 298-314
[4]. R. Prakash and M. Singhal. "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems". IEEE Trans. on Paralleland Distributed System, pages 1035-1048,Oct. 1996.
[5]. Koo R, Toueg S " Checkpointing and rollback recovery for distributed systems".IEEE Trans.Software Eng. SE-13: 23-31,1987
[6]. G. Cao and M. Singhal. "On impossibility of Min-Process and Non-Blocking Checkpointing and An Efficient Checkpointing algorithm for mobile computing Systems". OSU Technical Report #OSU-CISRC-9/97-TR44, 1997.
[7]. Prakash R. and Singhal M. "Maximal Global Snapshot with concurrent initiators," Proc. Sixth IEEE Symp. Parallel and Distributed Processing, pp.344-351, Oct.1994
[8]. Bidyut Gupta, S.Rahimi and Z.Lui. "A New High Performance Checkpointing Approach for Mobile Computing Systems". IJCSNS International Journal of Computer Science and Network Security, Vol.6 No.5B, May 2006.
[9]. Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.
[10]. Ch.D.V. Subba Rao and M.M. Naidu. "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging"
[11]. J.L.Kim and T.Park. "An efficient protocol for checkpointing recovery in Distributed Systems" IEEE Transaction On Parallel and Distributed Systems,4(8):pp.955-960, Aug 1993.
[12]. Yanping Gao, Changhui Deng, Yandong Che. "An Adaptive Index-Based Algorithm using Time-Coordination in Mobile Computing". International Symposiums on Information Processing, 2008.
[13]. Kanmani - Anitha - Ganesan ."Coordinated Checkpointing with Avalanche Avoidance for Distributed Mobile Computing System." International Conference on Computational Intelligence and Multimedia Applications 2007.
[14]. Ajay D Kshemkalyani: "A symmetric O(n log n) message distributed snapshot algorithm for large scale systems" IEEE, 2010, pp 1-4
[15]. Ajay D Kshemkalyani " Fast and message efficient global snapshot algorithms for large scale distributed systems IEEE 2010. Page(s): 1281 – 1289.
[16]. Silva L, Silva J 1992 Global checkpointing for distributed programs. Proc. IEEE 11th Symp. On Reliable Distributed Syst. pp 155-162.
[17]. Wang, Y.M., Fuchs, W.K.: Lazy checkpoint coordination for bounding rollback propagation. In: Proceedings of IEEE Symposium on Reliable Distributed Systems, pp. 78–85 (1993).
[18]. Kumar, P., Garg, R.: Soft Checkpointing Based Hybrid Synchronous Checkpointing Protocol for Mobile Distributed Systems. International Journal of Distributed Systems and Technologies 2(1), 1–13 (2011)
[19]. Venkatesan S 1993 Message-optimal incremental snapshots J. Comput. Sofnvare Engineering 1 211-31.
[20]. Rahul Garg, Vijay K Garg, Yogish sabharwal "Scalable algorithms for global snapshots in distributed systems" ACM 2006.
[21]. Kumar and Khunteta "A Minimum-Process Coordinated Check pointing Protocol For Mobile Distributed System" IJCSE, Vol. 02, No. 04, 2010, 1314-1326.
[22]. Gupta and Kumar "Review of Some Checkpointing Algorithms for Distributed and Mobile Systems" CNSA 2011, CCIS 196, pp. 167–177, 2011.
[23]. R. Tuli,P. Kumar," Minimum process coordinated Checkpointing scheme for ad hoc Networks", International Journal on AdHoc Networking Systems (IJANS) Vol. 1, No. 2, October 2011 ,pp-51-63.
[24]. M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.
[25]. Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", Ph.D. Thesis, Rutgers University, 1995.
[26]. David R. Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems, Vol. 7, NO.3, pp 404-425, July 1985.
[27]. 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.