Checkpointing Algorithms for Distributed Systems

Parveen Kumar[1], Richa Setiya[2], and Poonam Gahlan[2]
[1]*Dept. of Computer Science, APIIT, Panipat.*
[2]*Department of Computer Sc & Engg, Israna, Panipat*
[swastikaarya83@gmail.com]

***Abstract:-* 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 presents the review of the algorithms which have been reported in the literature for checkpointing parallel/distributed systems.**

Keywords: Checkpointing algorithms; parallel & distributed computing; shared memory systems; rollback recovery; fault-tolerant systems**.**

1. INTRODUCTION

Systems with more than one processor are known as multiprocessor systems. As the number of processors increase the probability of any one processor failing is high. Checkpointing, however, is more difficult in multiprocessors as compared to uniprocessors. This is due to the fact that in multiprocessors there are multiple streams of execution and there is no global clock. The absence of a global clock makes it difficult to initiate checkpoints in all the streams of execution at the same time instance. We have to pick one checkpoint from each stream in such a way that the set of these checkpoints are "concurrent". The concept of concurrency is defined based on the "happens before" relation defined by Lamport [6].

A distributed system is a collection of processes that communicate with each other by exchanging messages. A mobile distributed computing system is a distributed system where some of the processes are running on mobile hosts (MHs). The term "mobile" implies able to move while retaining its network connections. A host that can move while retaining its network connections is an

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

1.1 Uncoordinated Checkpointing

In uncoordinated or independent checkpointing, processes do not coordinate their checkpointing activity and each process records its local checkpoint independently.It eliminates coordination overhead all together and forms a consistent global state on recovery after a fault [12]. After a failure, consistent global checkpoint is established by tracking the dependencies. It may require cascaded rollbacks that may lead to the initial state due to domino-effect [13]. 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.

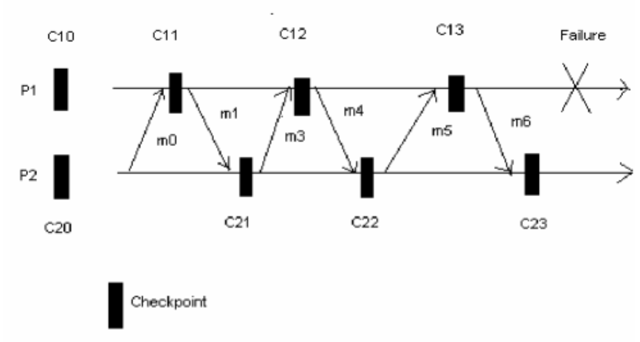

Figure 2. Domino-effect

The main disadvantage of this approach is the domino-effect [Figure 2]. 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 [2]. 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.

1.2 Coordinated Checkpointing

In coordinated or synchronous checkpointing, processes take checkpoints in such manner that the resulting global state is consistent. Mostly it follows two-phase commit structure [13],[14],[15]. 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. A permanent checkpoint can not be undone. It guarantees that the computation needed to reach the checkpointed state will not be repeated. A tentative checkpoint, however, can be undone or changed to be a permanent checkpoint. The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, as mentioned above, some blocking of processes takes place during checkpointing [13]. In non-blocking algorithms, no blocking of processes is required for checkpointing [14],[15].

1.3 Quasi-synchronous or communication induced checkpointing

Communication-induced checkpointing avoids the domino-effect without requiring all checkpoints to be coordinated [19]. 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 .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.

1.4 Message logging based checkpointing protocols

Message-logging protocols (for example [16],[17],[18], are popular for building systems that can tolerate process crash failures. Message logging and checkpointing can be 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. No coordination is required between the checkpointing of different processes or between message logging and checkpointing. 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 [15]. Thus, message- logging protocols guarantee that upon recovery, no process is an orphan.

2. CHECKPOINTING ALGORITMS FOR MESSAGE-PASSING SYSTEMS

Chandy & Lamport (1985) proposed a global snapshot algorithm for distributed systems. We observe that every checkpointing algorithm proposed for message-passing (MP)systems uses Chandy & Lamport's (1985) algorithm as the base. We show that most of the algorithms proposed in the literature for checkpointing MP systems may be derived by relaxing various assumptions made by them and by modifying the way each step is carried out. As per Chandy and Lamport's model, a distributed system consists of a finite set of processors and a finite set of channels[6].

2.1 Algorithm: The global state is constructed by coordinating all the processors and logging the channel states at the time of checkpointing. Special messages called markers are used for coordination and for identifying the messages originating at different checkpoint intervals. The algorithm is initiated by a centralised node. The steps followed after a checkpoint initiation, however, are the same in all the nodes except that a centralised node initiates checkpoint on its own and the other nodes initiate checkpoints as soon as they receive a marker. The steps are as below.

```
(1) Save the local context in 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 is received along that channel[6].
```

2.2 Modifications of Chandy and Lamport's algorithm

Each step of the CL algorithm can be modified to accommodate some improvements in the basic global snapshot algorithm. In step one, a node saves its context in stable storage. The overhead associated with step one is context-saving overhead. The objective of saving the context in stable storage is to ensure its availability after a node failure. The overhead of context saving is proportional to the size of the context and the time taken to access the stable storage. Context-saving overhead can thus be reduced by (a) minimising the context size, and (b) overlapping context saving with computation.

In step two, markers are sent along all the 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 the current checkpoint interval.

At the time of checkpointing the centralised node informs all the nodes to initiate checkpoints through this marker message. CL algorithm sends markers along every channel to inform the nodes to log all transit messages onto stable storage .Checkpointing can be coordinated without using markers by sending with regular messages a header which has the checkpoint interval number in which the message originated. The simplest would be a one-bit header, which toggles between one and zero indicating the consecutive checkpoint intervals[21]. Note that the marker overhead has now become header overhead; overhead due to appending headers with regular messages. When a message is received with a header value different from that of the receiving node, either a new checkpoint is initiated or the message is logged depending on whether the message is an orphan message or a missing message. This one-bit header complicates checkpoint initiation when out-of-sequence messages are encountered. Message sequence numbers along with checkpoint interval number in the message header can help in controlling the number of checkpoints along with logging of missing messages and elimination of orphan messages[13]. The cost of this approach is the size of the header for maintaining the message sequence numbers and checkpoint interval number. When nodes initiate checkpoints on their own, it is called distributed checkpointing. Independent checkpointing eliminates coordination overhead at runtime and forms a consistent global state only when it is needed, i.e only at recovery time.When there is no coordination, nodes should be able to initiate checkpoints independently on their own. The advantages of this independent checkpointing are that 1) coordination and thereby the use of markers is eliminated; .2) nodes can initiate checkpoints at their convenience without being forced to initiate by the receipt of marker messages. The disadvantage is the maintenance of multiple checkpoints and message logs.Yet another mode of coordination is to synchronise the clocks and initiate the checkpoints approximately at the same time in all the nodes [7]. To account for the differences in the clock values, message sending can either be delayed during checkpointing or headers can be used with messages. Step three of the CL algorithm allows regular processing to proceed without waiting for the channel state recording and consequently the checkpoint operation to be completed. This is a good way of reducing the intrusion of a checkpointing algorithm but a better approach would be to overlap the context-saving process with regular computation. Step four of CL algorithm logs those messages which cannot be generated at recovery time.The purpose served by markers in identifying these messages can also be fulfilled by headers and this was mentioned while discussing step two.

3. CONCLUSION

Coordinated checkpointing generally simplifies recovery and garbage collection, and yields good performance in practice. At the other end of the spectrum,uncoordinated 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. Recent studies, however, have shown that the non-deterministic nature of these protocols complicates garbage collection and degrades performance.Causal logging reduces the overhead while still preserving the properties of fast output commit and orphan-free recovery.

REFERENCES

[1] Koo R, Toueg S " Checkpointing and rollback recovery for distributed systems".*IEEE Trans.Software Eng. SE*-13: 23-31,1987

[2] Zomaya A Y H " Parallel and distributed computing handbook*" (New York: McGraw-Hill)*,1996

[3] Siewiorek D P, Swarz S "The theory and practice of reliable system design*" (Cambridge, MA: Digital Press)*,1982

[4] Li K, Naughton J F, Plank S " Checkpointing multicomputer applications*". Proc. IEEE Conf. on Reliable Distributed Syst.* pp 2-11,1991

[5] Wang Y-M, Chung P-Y, Lin I-J, Fuchs W K "Checkpoint space reclamation for uncoordinated checkpointing in message passing systems". *IEEE Trans. Parallel Distributed Syst.* 6: 546-554,1995

[6] Chandy K M, Ramamoorthy C V "Rollback and recovery strategies for computer programs". *IEEE Trans. Comput. C*-21: 546-556,1972

[7] Cristian F, Jahanian F *"* A timestamp-based checkpointing protocol for long-lived distributed computations". *Proc. IEEE Conf. on Reliable Distributed Syst.* pp 12-20,1991

[8] Tong Z, Kain R Y, Tsai W T " Rollback recovery in distributed systems using loosely synchronized clocks".*IEEE Trans. Parallel Distributed Syst.* 3: 246-251,1992

[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] Acharya A., "Structuring Distributed Algorithms and Services for networks with Mobile Hosts", *Ph.D. Thesis*, *Rutgers University*, 1995.

[11] Badrinath B. R, Acharya A., T. Imielinski "Structuring Distributed Algorithms for Mobile Hosts", *Proc. 14th Int. Conf. Distributed Computing Systems*, June1994.

[12] 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.

[13] 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.

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

[15] 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.

[16] Alvisi, Lorenzo and Marzullo, Keith,"Message Logging: Pessimistic, Optimistic, Causal, and Optimal", *IEEE Transactions on Software Engineering*, Vol. 24, No. 2, February 1998, pp. 149-159.

[17] 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.

[18] 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. *of the International Symposium on Fault-Tolerant-Computing Systems*, pp. 68-77,June 1997.

[19] 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.

[20] Silva L, Silva J “Global checkpointing for distributed programs*”. Proc. IEEE 11th Symp. On Reliable Distributed Syst.* pp 155-162,1992

[21] Lai T H, Yang T H “On distributed snapshots”.*Inf. Process. Lett.* 25: 153-158,1987

[22] Leu P-J, Bhargava B “Concurrent robust checkpointing and recovery in distributed systems”.*Proc. Int. Conf. on Data Engineering* pp 154-163,1988

0-107,1996

[23] Ahmed R E, Frazier R C, Marinos P N “Cache aided rollback error recovery (CARER) algorithms for shared memory multiprocessor systems*”. Proc. IEEE 20th Int. Symp. on Fault Tolerant Computing* pp 82-88,1990

[24] Wu K-L, Fuchs W K, Patel J H “Cache based error recovery for shared memory multiprocessor systems”. *Proc. Int. Conf. on Parallel Processing* pp I159-I166,1989

[25] Tam V-O, Hsu M “ Fast recovery in distributed shared virtual memory systems*”. Proc. IEEE 10th Int. Conf. on Distributed Computing Syst.* pp 38-45,1990

[26] Kalaiselvi S, Rajaraman V “Task graph based checkpointing in parallel/distributed systems”.*J.Parallel Distributed Comput. (submitted),*2000

[27] Manivannan D, Singhal M “A low-overhead recovery technique using quasi synchronous checkpointing*”. Proc. IEEE Int. Conf. on Distributed Computing Syst.* pp 10