Review of Checkpointing Algorithms in Distributed Systems

Poonam Gahlan[1], Parveen Kumar[2]

[1]Department of Computer Sc & Engg,Israna,Panipat
[2]Department of Computer Science & Engineering, Meerut Institute of Engineering & Technology, Meerut India
pk223475@yahoo.com

***Abstract*:** Checkpointing is the process of saving the status information. 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. Mobile computing raises many new issues such as lack of stable storage, low bandwidth of wireless channel, high mobility, and limited battery life. Coordinated checkpointing is an attractive approach for transparently adding fault tolerance to distributed applications since it avoids domino effects and minimizes the stable storage requirement. This paper presents the review of the algorithms, which have been reported in the literature for checkpointing. This paper also covers backward error recovery techniques for distributed systems specially the distributed mobile systems.

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

1. 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. The messages are delivered after an arbitrary delay. Parallel computing with clusters of workstations (cluster computing) is being used extensively as they are cost-effective and scalable, and are able to meet the demands of high performance computing. Increase in the number of components in such systems increases the failure probability. There are mainly two kinds of faults: permanent and transient. Permanent faults are caused by permanent damage to one or more components 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. Hence it is necessary to provide fault tolerance particularly for transient failures in parallel computers. Fault-tolerant techniques enable a system to perform tasks in the presence of faults. Fault tolerance involves fault detection, fault location, fault containment and fault recovery.

2. Inherent Limitations of Distributed System and motivation

In a distributed system, there exists no system wide common clock (global clock). In other words, the notion of global time does not exist. A reader might think that this problem can be easily solved by either having a clock common to all the computers (processes) in the system or having synchronized clocks, one at each computer.

Since the computers in a distributed system do not share common memory, an up-to–date state of the entire system is not available to any individual process. Up –to-date state of the system is necessary for reasoning about the system's behavior, debugging, recovering from failures.

The motivation for using a distributed system is some or all of the following requirements:

1. **Inherently distributed computations** In many applications such as money transfer in banking, or reaching consensus among parties that are geographically distant, the computation is inherently distributed.

2. **Resource sharing** Resources such as peripherals, complete data sets in databases, special libraries, as well as data (variable/files) cannot be fully replicated at all the sites because it is often neither practical nor cost-effective. Further, they cannot be placed at a single site because access to that site might prove to be a bottleneck. Therefore, such resources are typically distributed across the system.

3. **Access to geographically remote data and resources** In many scenarios, the data cannot be replicated at every site participating in the distributed execution because it may be too large or too sensitive to be replicated. For example, payroll data within a multinational corporation is both too large and too sensitive to be replicated at every branch office/site. It is therefore stored at a central server which can be queried by branch offices. Similarly, special resources such as supercomputers exist only in certain locations, and to access such supercomputers, users need to log in remotely.

4. **Enhanced reliability** A distributed system has the inherent potential to provide increased reliability Reliability entails several aspects:

• availability, i.e., the resource should be accessible at all times;

• integrity, i.e., the value/state of the resource should be correct, in the face of concurrent access from multiple processors, as per the semantics expected by the application;

• fault-tolerance, i.e., the ability to recover from system failures, where such failures may be defined to occur in one of many failure models.

5. **Increased performance/cost ratio** By resource sharing and accessing geographically remote data and resources, the performance/cost ratio is increased. Such a configuration provides a better performance/cost ratio than using special parallel machines. This is particularly true of the NOW configuration.

6. **Scalability** As the processors are usually connected by a wide-area network, adding more processors does not pose a direct bottleneck for the communication network.

3. Literature survey

3.1 Guohong Cao and Mukesh Singhal(1997)[11]

proposed a min-process checkpointing algorithm 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, we develop an algorithm that relaxes the non-blocking condition; that is, it is a

In this, they introduce 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.

The Basic Idea behind Non-blocking Algorithms

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

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. After the initiator has received positive replies from all relevant processes.

second phase, if the initiator learns that all 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.

3.2 Lalit Kumar Awasthi and P.Kumar (2007)[5] proposed algorithm is based on keeping track of direct dependencies of processes. Initiator MSS collects the direct dependency vectors of all processes, computes the tentative minimum set (minimum set or its subset), and sends the checkpoint request along with the tentative minimum set to all MSSs. This step is taken to reduce the time to collect the coordinated checkpoint. It will also reduce the number of useless checkpoints and the blocking of the processes. Suppose, during the execution of the checkpointing 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 [18]. 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 minimise the number of useless checkpoints and blocking of the 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 duplicate checkpoint requests.

3.3 Silva and Silva [13] 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 . When a process receives a computational message with the high 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 program is allowed to initiate the checkpoint operation, the network may be flooded with control messages and process 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 the checkpoint 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.

3.4 Koo-Toueg's[8] proposed a minimum process blocking checkpointing algorithm for distributed systems. 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 process 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 process 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 messages after the checkpoint is permissible.

3.5 Xu and Netzer [12] introduced the notion of Zigzag paths, a generalization of Lamport's happened-before relation 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 shown that a set of checkpoints can belong to the same consistent global snapshot iff no zigzag path exists from a checkpoint to any other.

3.6 Coa and Singhal[2] 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 depend. 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.

3.7 Elnozahy and Zwaenepoel[10] 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.

3.8 Prakash-Singhal algorithm [7] was the first algorithm to combine these two approaches. More specifically, it forces only a minimum number of processes to take checkpoints and does not block the underlying computation during checkpointing. Prakash Singhal algorithm forces only part of processes to take checkpoints, the csn of some processes may be out-of-date, and may not be able to avoid inconsistencies. Prakash-Singhal algorithm attempts to solve this problem by having each process maintains an array to save the csn, where $csn_i[i]$ has been the expected csn of P*i*. Note that P*i's* $csn_j[i]$ may be different from P_j 's csn_j [i] if there is no communication between P_i and P_j for several checkpoint intervals. By using csn and the initiator identification number, they claim that their non-blocking algorithm can avoid inconsistencies and minimize the number of checkpoints during checkpointing.

3.9 Kalaiselvi VI and V Rajaraman (2000)[3]

Chandy & Lamport proposed a global snapshot algorithm for distributed systems. They observe that every checkpointing algorithm proposed for message-passing (MP) systems uses Chandy & Lamport's [1] algorithm as the base. They 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.

Chandy and Lamport's algorithm:

Chandy and Lamport's CL algorithm is based on the following assumptions.

- The distributed system has a finite number of processors and a finite number of channels.
- The processors communicate with each other by exchanging messages through communication channels.
- The channels are fault-free.
- 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 and not yet
- received by the destination node from that channel.
- Buffers are of infinite capacity.
- Termination of the algorithm is ensured by fault-free communication.

Algorithm: coordinating all the processors and logging the channel states at the time of checkpointing construct the global state. Special messages called markers are used for coordination and for identifying the messages originating at different checkpoint intervals. A centralised node initiates the algorithm. 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.

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. (a) Minimizing the context size, and (b) overlapping context saving with computation can thus reduce context-saving overhead.

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 centralized 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. It is not necessary to send markers along all the channels as they may be safely eliminated along those channels in which there was no message exchange between the previous and the current checkpoint [4], [6]. Coordination through markers can also be achieved in two phases by delaying the message transmission between the two phases. 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. 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. The cost of this approach is the size of the header for maintaining the message sequence numbers and checkpoint interval umber.When nodes initiate checkpoints on their own, it is called distributed checkpointing. If checkpoint initiation completely depends on the header of regular messages, those nodes, which have not communicated, with other nodes between consecutive checkpoints cannot participate in a global checkpoint. One can also use markers just to inform about checkpoints; markers take care of coordination and headers take care of message logging [6]. In all the schemes mentioned above, coordination is achieved at runtime and a consistent global state is always maintained in stable storage.The next major alternative called independent checkpointing eliminates coordination overhead at runtime and forms a consistent global state only when it is needed, i.e. only at recovery time. Instead of coordinating the nodes during every global checkpoint, nodes can be coordinated once at recovery time to form a consistent global state. When there is no coordination, nodes should be able to initiate checkpoints independently on their own. To form a consistent global state at recovery time, nodes have to maintain multiple checkpoints and messages in stable storage. The advantages of this independent checkpointing are that

*(i)*coordination and thereby the use of markers is eliminated;
(ii) 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. Multiple checkpoints occupy more space and garbage collection algorithms can be run periodically to reclaim the space occupied by unwanted checkpoints. Consistent global state is constructed periodically and all the checkpoints which do not belong to the recovery line are declared unwanted checkpoints. Though special messages are used for identifying the recovery line, the frequency of usage is lower when compared with a coordinated algorithm based on markers. The other significant overhead in independent checkpointing is due to logging of messages (logging overhead) since it has to log all the messages received. Pessimistic logging approach has the advantage of faster recovery since it logs a message as and when it is received (Borg et al 1989). By grouping the messages over a period and logging them once in a while optimistic logging approaches reduce the stable storage access overhead. If sufficient messages are not logged, multiple rollbacks are possible in optimistic logging schemes. One can also send sufficient information with regular messages so that messages can be logged selectively thereby reducing the message logging overhead. Optimistic schemes need a complicated recovery procedure. The advantages of pessimistic and optimistic schemes can be combined to achieve minimum logging overhead with faster recovery. Further modification of independent checkpointing algorithm is possible depending on where a message is logged; at places other than the receiver. The advantage is that the messages need not be logged onto stable storage. Yet another mode of coordination is to synchronize the clocks and initiate the checkpoints approximately at the same time in all the nodes. 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.

4. Conclusion

A majority of Checkpointing algorithms are based on the

seminal article by Chandy & Lamport (1985.More recent published work attempts to minimise the context-saving overhead. 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. A non-blocking checkpointing algorithm does not require any process to suspend its underlying computation. When processes do not suspend their computation, it is possible for a process to receive a computation message from another process, which is already running in a new checkpointing interval. If this situation is not properly dealt with, it may result in an inconsistency. However, these algorithms assume that a distinguished initiator decides when to take a checkpoint. Therefore, they suffer from the disadvantages of centralized algorithms. Most of non-blocking algorithms, use a checkpoint sequence number (csn) to avoid inconsistencies. This scheme works only when every process in the computation can receive each checkpoint request then increases its own csn.

References

[1] 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.

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

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

[5]. 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

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

[7] R. Prakash and M. Singhal. "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems". IEEE Trans. on Parallel and Distributed System**,** pages 1035-1048,Oct. 1996.

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

[9].Chandy K M, Lamport L 1985 Distributed snapshots: Determining global states of distributed systems. ACM Trans. Comput. Syst. 3: 63-75.

[10].Elnozahy E N, Zwaenepoel W 1994 On the use and implementation of message logging Proc.IEEE Int.Symp. on Fault Tolerant Computing pp 298-307.

[11]. 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.

[12]. Netzer R H B, Xu J 1995 Necessary and sufficient conditions for consistent global snapshots IEEE Trans. Parallel Distributed Syst. 6: 165-169.

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