A Review on Checkpoint Algorithms for Parallel and Distributed Computers

Pallvi Kaura¹, Dr. Mohita²
¹Research Scholar, INDIA
²Associate Professor, North West Institute of Engineering & Technology, INDIA

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. A large number of articles have been published in this area by relaxing the assumptions made in this paper and by extending it to minimise the overheads of coordination and context saving. Checkpointing for shared memory systems primarily extend cache coherence protocols to maintain a consistent memory. All of them assume that the main memory is safe for storing the context. Recently algorithms have been published for distributed shared memory systems, which extend the cache coherence protocols used in shared memory systems. They however also include methods for storing the status of distributed memory in stable storage. Most of the algorithms assume that there is no knowledge about the programs being executed. It is however felt that in development of parallel programs the user has to do a fair amount of work in distributing tasks and this information can be effectively used to simplify checkpointing and rollback and recovery.

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

I. INTRODUCTION

To provide fault tolerance it is essential to understand the nature of the faults that occur in these systems. 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.

In the application-system software level, checkpointing techniques are used to provide fault tolerance. It is easier and more cost effective to provide software fault tolerance solutions than hardware solutions to cope with transient failures. Thus, checkpointing is an important technique to ensure software fault tolerance.

1.1 Applications

It is use to recover from failures, checkpointing is also used in debugging distributed programs and migrating processes in a multiprocessor system. In debugging distributed programs state changes of a process during execution are monitored at various time instances. Checkpoints assist in such monitoring. To balance the load of processors in a distributed system, processes are moved from heavily loaded processors to lightly loaded ones. Checkpointing a process periodically provides the information necessary to move it from one processor to another.

Checkpoints in Single Processor Systems

In a Parallel System transactions are governed by a single clock, Which provides a total ordering of events with respect to this clock. Thus checkpointing may be performed at a specified clock time by stopping the execution of the process and saving the state of the process in a stable storage. When an error is detected all the events after the last checkpoint are repeated.

Checkpoints in Distributed Systems

Systems with more than one processor are known as multiprocessor systems. We use distributed systems, parallel systems and multiprocessor systems interchangeably referring to all of them as multiprocessor systems. As the number of processors increase the probability of any one processor failing is high.

Checkpointing and rollback recovery are particularly useful in such situations. Check-pointing,
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". Such a set of checkpoints permits a consistent rollback recovery.

II. CHECKPOINTING ALGORITHMS FOR MESSAGE-PASSING SYSTEMS

Chandy&Lamport (1985) proposed a global snapshot algorithm for distributed systems. It is observed that every checkpointing algorithm proposed for message-passing (MP) systems uses Chandy&Lamport's (1985) algorithm as the base. As per Chandy and Lamport's model, a distributed system consists of a finite set of processors and a finite set of channels (for a detailed description of the model, refer Chandy&Lamport's 1985) algorithm is ensured by fault-free communication.

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: 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 as stable**
2. **Send markers along channel**
3. **Continue regular computation**
4. **Save incoming messages in channel i until a marker is received along that channel**

III. CHECKPOINTING ALGORITHMS FOR SHARED-MEMORY SYSTEMS

Shared-memory (SM) systems have a global address space and nodes communicate using shared variables. To reduce the memory latency, shared-memory systems use cache memory which in turn requires coherence of caches. Cache coherence protocols (Archibald & Baer 1986) help in maintaining a consistent memory. Checkpointing algorithms are incorporated as part of the cache protocols because at the time of checkpointing all cache lines must be updated and at the time of recovery no cache line should be updated more than once. The shared variables of a SM system are equivalent to the messages of a message passing system as they introduce dependency among the processes in the same way the messages introduce dependency among the processes in a message passing system.

3:1 Cache-based checkpointing algorithms

The basic checkpointing algorithm for a SM system is given below.

1. At the time of checkpointing, make the main memory consistent using the cache coherence protocols.
2. Save the process contexts in the memory.
3. Save the global state in secondary storage.

A checkpoint can be taken whenever the memory is consistent and the state of all processes are available. Existing checkpointing algorithms for shared-memory systems initiate checkpoints based on the cache line modifications (cache-based algorithms). In the simplest case, a node initiates a global checkpoint whenever the number of dirty cache lines exceed a threshold (Ahmed et al 1990). This requires all the nodes to participate in the global checkpoint. Limiting the participation to only those nodes which are dependent on each other improves the algorithm. Another alternative is to initiate a global checkpoint whenever the effect of a modified cache is made visible to other nodes. The above three approaches were proposed by Ahmed et al (1990).

The cache-based recovery algorithm (Wu et al 1989) assumes the presence of a special memory called recovery stack, to reduce the time taken for context saving, when a cache line is updated, it is written onto recovery stack instead of the main memory. The main memory is updated either when the recovery stack is full or when a new checkpoint is initiated. The size of the recovery stack and the pattern of interaction among the processes determine the frequency of checkpointing.

One important point to observe in the cache-based algorithms is that the context is not saved in stable storage. The algorithms are meant only for soft error recovery (Ahmed et al 1990). The context is either maintained in a special memory or in the main memory. It is assumed that the memory is safe and the state of the processors can be restored from memory. The advantages of soft recovery approach are: (a) time taken for checkpointing is considerably less than the...
checkpointing which moves the state to stable store, (b) large number of checkpoints can be initiated and the time interval between consecutive check-points can be small. The disadvantage is that memory failure will require the system to restart from the beginning, making the efforts expended on checkpointing useless. In this kind of soft error recovery algorithm, the checkpointing overhead is mainly due to the large number of checkpoints. Moreover, when there is no control over the number of check-points, it is not advisable to take a checkpointing approach that saves context in stable storage. There should be assurance that the time between consecutive checkpoints is large enough for saving the context in stable storage. From the above observation it is clear that the frequency of checkpointing cannot be predicted in any cache-based algorithm and is influenced significantly by the pattern of interaction among the nodes.

IV. CHECKPOINTING ALGORITHMS FOR DISTRIBUTED SHARED-MEMORY SYSTEMS

Distributed shared memory (DSM) systems have global address space like shared-memory systems but the memory is distributed across all the nodes. It is a software layer that provides the appearance of a shared-memory system to the user and internally communicates through messages. Caches are present to minimise latency in data access and the programming paradigm is shared-memory paradigm. Since DSM systems are similar to SM systems, the checkpointing algorithms for DSM systems should concentrate on making the memory consistent at the time of checkpointing. DSM systems cannot, however, use checkpointing algorithms identical to SM systems because a node failure makes a portion of the global memory not available, unlike an SM system where memory is separate and is assumed to be safe. To tolerate node failures, checkpoints should be maintained in stable storage like MP systems.

Considering the facts mentioned above, a basic DSM checkpointing algorithm is given below.

1. At the time of checkpointing, make the distributed main memory consistent through the memory management protocols.
2. Save the process contexts in the memory. Save the global state in secondary storage.

By examining the few algorithms available in the literature for DSM systems, it is clear that the approaches are extensions of cache-based algorithms because the dependency among the nodes is created through sharing of memory. We can call them memory-based algorithms because they are incorporated as part of memory coherence protocols. Note that these algorithms resort to techniques for overlapping the context-saving process with computation because the time taken to write the context in stable storage is significant.

V. CLASSIFICATION OF CHECKPOINTING ALGORITHMS

5:1 Classification tree

The classification tree is given in figure 1. The first level of classification is based on the availability of a global clock. Absence of a global clock has led to a variety of checkpointing algorithms in multiprocessor systems. In both uniprocessor and multiprocessor systems two major approaches known as static and dynamic approaches are used to identify checkpointing locations. In uniprocessors, static approaches identify the checkpointing locations prior to program execution using either a task graph of the program (Chandy & Ramamoorthy 1972) or by the compiler analysing the program (Li & Fuchs 1990) the dynamic approach identifies the checkpointing locations at runtime.

In multiprocessor systems, the static approach has not been widely used except for a recent algorithm based on task graphs (Kalaiselvi & Rajaraman 2000). Almost all the algorithms we have seen for distributed systems are dynamic. Dynamic algorithms for multiprocessors are classified based on the architecture of the system as it determines the way dependencies are introduced among the communicating processes which in turn influence the way checkpointing has to be carried out. Shared memory systems use cache-based algorithms and they are meant only for soft error recovery. Distributed shared memory systems use memory-based schemes that can withstand node failures.

In message-passing systems, the major difference in algorithms is based on whether coordination is done at runtime or at recovery time. Coordinated approaches coordinate the nodes and form a consistent state at runtime, whereas independent algorithms form a consistent state only at recovery time. The recent approach is to coordinate the checkpoints partially and is called quasi-synchronous approach (Manivannan & Singhal 1996). In coordinated algorithms, coordination is achieved through markers, headers or both. The main distinction between approaches used in independent schemes is the method used to log messages. They follow pessimistic, optimistic or a combination of pessimistic and optimistic logging methods. Optimistic methods can further be classified based on where the messages are logged: sender-based logging or receiver-based logging.
VI. CONCLUSIONS

In this review paper on checkpointing algorithms has shown, a large number of papers have been published on checkpointing message-passing distributed computers. A majority of these algorithms are based on the seminal article by Chandy & Lamport (1985) and have been obtained by relaxing many of the assumptions made by them; the main aim of improving the earlier extensions of the Chandy & Lamport (1985) algorithms was to minimise the overhead of coordination between processes in a multiprocessor system.

A smaller number of algorithms have been proposed to checkpoint shared-memory multiprocessors. These algorithms primarily extend cache coherence protocols to maintain a consistent memory. These algorithms assume the main memory to be safe and do not save context in disk.

More recently, algorithms have been proposed for distributed shared-memory systems. In these systems also maintenance of cache coherence of the logical global memory is important for checkpoints. As the physical memory is distributed it is necessary to save main memory contents in the disk. Thus context saving overhead is higher when compared to shared-memory systems.

We also see that most of the algorithms assume no prior knowledge on the structure of programs meant for execution on multiprocessors. In practice, considerable information about such programs is available. We suggest that use of this knowledge by checkpointing algorithms can considerably reduce the coordination and context-saving overheads of such algorithms.

REFERENCES