A Comparison between Different Checkpoint Schemes with Advantages and Disadvantages

Manoj Kumar Abhishek Choudhary Vikas Kumar

ABSTRACT
It is known that check pointing and rollback recovery are widely used techniques that allow a distributed computing to progress in spite of a failure. There are two fundamental approaches for check pointing and recovery. One is asynchronous approach, process take their checkpoints independently. So, taking checkpoints is very simple but due to absence of a recent consistent global checkpoint which may cause a rollback of computation. Synchronous check pointing approach assumes that a single process other than the application process invokes the check pointing algorithm periodically to determine a consistent global checkpoint. Various flavors of these two techniques, their mechanisms, advantages and drawbacks have been discussed in detail. Besides an exhaustive study of the implementation issues are also included. Lastly, some open issues have been addressed and certain solutions have been proposed by the author.

Keywords
Recovery, Rollback recovery, Synchronous, Asynchronous, Checkpoint, cp.

INTRODUCTION
Checkpoint and recovery protocols are commonly used in distributed applications for providing fault tolerance. Check pointing is one of the fault-tolerant techniques to restore faults and to restart job fast. Check pointing-based recovery is a well-known method to allow the current faulty state of a fail-stop processors system [1] to be rolled back to a globally consistent state of the system recorded on stable storage before its failure [2, 3, 4]. It is reasonable to say that the major source of overhead in check pointing schemes is the stable storage latency. Communication overhead becomes a minor source of overhead as the latency of network communication decreases. In this scenario, the coordinated checkpoint becomes worthy since it requires less accesses to stable storage then uncoordinated checkpoints. Furthermore, in practice, the low overhead gain on uncoordinated check pointing do not justify neither the complexities of finding the recovery line after failure and performing the garbage collection nor the high demand for storage space caused by multiple checkpoints of each process. Checkpoint and recovery protocols are commonly used in distributed applications for providing fault tolerance. A distributed system may require taking checkpoints from time to time to keep it free of arbitrary failures. In case of failure, the systems will rollback to checkpoints where global consistency is preserved. Check pointing is one of the fault-tolerant techniques to restore faults and to restart job fast. The performance of a checkpoint and recovery protocol is judged by the amount of computation it can save against the amount of overhead it incurs. This performance depends on different system and application characteristics as well as protocol specific parameters. Hence, no single checkpoint and recovery protocol works equally well for all applications. Given a distributed application and a system it will run on, it is important to choose a protocol that will give the best performance for that system and application. In this thesis, a Recover Algorithm in conjunction with Check pointing algorithm which is efficient, decentralized and cost effective and suitable for cluster federation has been proposed. During normal computation message transmission, dependency information among clusters is recorded in the corresponding cluster head processes. When a check pointing procedure begins, the initiator from a cluster concurrently informs all the cluster head processes which further multicast messages to currently active processes in their corresponding clusters thus resulting in reduced transmission delay, communication cost, better bandwidth utilization and faster speed of execution. Quantitative analysis shows that proposed algorithm outperforms other check pointing schemes and can provide a better system performance for cluster federation. Considerable research has been devoted to checkpoint-based backward recovery schemes [9, 10]. There have also been techniques proposed which combine replication with voting and checkpoint rollback recovery. Recovery from transient failures is one of the prime issues in the context of distributed systems. What is desirable is to have transparent yet efficient techniques to achieve the same. This report contains a comprehensive study of the existing techniques, namely Checkpoint-based recovery and Log-based recovery. Various flavors of these two techniques, their mechanisms, advantages and drawbacks have been discussed in detail. Besides an exhaustive study of the implementation issues is also included. Lastly, some open issues have been addressed and certain solutions have been proposed by the author.

1. Phases of Check pointing:

Check pointing has two phases:
• Saving a checkpoint
• Checkpoint recovery following the failure.

To save a checkpoint, the memory and system, necessary to recover from a failure is sent to storage. Checkpoint recovery involves restoring the system state and memory from the checkpoint and restarting the computation from the checkpoint stored [5].

1.1 Types of Check pointing:

There are following types of check pointing:
A) Disk based check pointing
B) Disk less check pointing
C) Double check pointing

A) Disk Based Check pointing:

In checkpoint based methods, the state of the computation as a checkpoint is periodically saved to a stable storage, which is not subject to failures. When a failure occurs the computation is restarted from one of these previously saved states. According to the type of coordination between different processes while taking checkpoints, checkpoint-based methods can be broadly classified into three categories:

i) Uncoordinated check pointing or asynchronous check pointing
ii) Coordinated check pointing or synchronous check pointing
iii) Communication-induced or Quasi-Synchronous or Hybrid Check pointing

B) Diskless Check pointing:

It is a technique for distributed system with memory and processor redundancy. It requires two extra processors for storing parity as well as standby. Process migration feature has ability to save a process image. The process can be resumed on the new node without having to kill the entire application and start it over again. It has memory or disk space .In order to restore the process image after a failure, a new processor has to be available to replace the crashed processor. This requires a pool of standby processors for multiple unexpected failures [6].

The comparison between disk based and disk less check pointing for distributed and parallel system in certain parameter is described in table 1 [6].

Table 1: On Disk and Disk less check pointing for distributed system

Parameter	Disk Based	Diskless
Latency time	High	Low
CPU Overhead	High	High
Memory Requirement	Low	High
Stable Storage Requirement	High	Low
Toleration of Wholesale Failure	Yes	No
Reliability	High	Low
Efficiency	Low	High
Addition Hardware	Not Required	Additional Processors
Portability	High	Low

C) Double Check pointing:

Double check pointing targets on relatively small memory footprint on very large number of processors when handles fault at a time, each checkpoint data would be stored to two different locations to ensure the availability of one checkpoint. In case, one is lost, other can be used since two buddy processors have identical checkpoints. It can be stored either in the memory or local disk of two processors. These are double in-memory check pointing and double in-disk check pointing schemes. This scheme stores checkpoint in a distributed fashion to avoid the network bottleneck to the central server [7]. The comparison between Disk-based and Memory-based Checkpoint in certain parameter is described in table 2.

- **Double In-memory Check pointing**

In this check pointing each process stores its data to memory of two different processors. It has faster memory accessing capability, low checkpoint overhead and faster restart to achieve better performance than disk-based checkpoint. But it will increase the memory overhead and initiate check pointing at a time when the memory footprint is small in the application. This can be applied to many scientific and engineering applications such as molecular dynamics simulations that are iterative.

Table 3: Comparison between different checkpoint schemes

Fault tolerant protocols	Double in Memory	Double in Disk
Shrink/Expand	Yes	Yes
Portability	Low	Low
Foolproof	NO	NO
Diskless	Yes	No, Local Disk
Halts job	No	No
Bottleneck	No	No
Require Backup Processors	Not Necessarily	Not Necessarily
Transparent Checkpoint	No	No
Synchronized Checkpoint	Yes	Yes
Automatic Restart	Yes	Yes

- **Double In-memory Check pointing**

In this check pointing each process stores its data to memory of two different processors. It has faster memory accessing capability, low checkpoint overhead and faster restart to achieve better performance than disk-based checkpoint. But it will increase the memory overhead and initiate check pointing at a time when the memory footprint is small in the application. This can be applied to many scientific and engineering applications such as molecular dynamics simulations that are iterative.

- **Double In-disk Check pointing**

It is useful for applications with very big memory footprint where checkpoints are stored on local scratch disk instead of in processor memory. Due to the duplicate copies of checkpoints it doesn't rely on reliable storage. It incurs higher disk overhead in check pointing but does not suffer from the dramatic increase in memory usage as in the double in-memory check pointing. Taking advantage of distributed local disks, it avoids the bottleneck to the central fileserver [8].

Comparisons between Different Checkpoint Schemes:

It is reasonable to say that the major source of overhead in check pointing schemes is the stable storage latency. Communication overhead becomes a minor source of overhead as the latency of network communication decreases. In this scenario, the coordinated checkpoint becomes worthy since it requires less accesses to stable storage then uncoordinated checkpoints. Furthermore, in practice, the low overhead gain on uncoordinated check pointing do not justify neither the complexities of finding the recovery line after failure and performing the garbage collection nor the high demand for storage space caused by multiple checkpoints of each process.

CIC protocol, in turn, does not scale well as the number of processes increase. The required amount of storage space is also difficult to predict because of the occurrence of forced checkpoint at random points of the application execution.

CP Methods Parameter	Uncoordinated CP	Coordinated CP	Communication Induced CP	Diskless CP	Double CP
Efficiency	High for small process	Low	Low	High	High
Performance	Low	Low	Low	Higher for distributed applications	Faster
Portability	High	High	High	Low	Low
Cost	High	low, negligible for low memory usage application	High	High	very high
Scalable	No	minimal	not scale for large number of processors	difficult to scale to large number of processors	Highly
Flexibility	Low	all processes save their states at the same time	process can be moved from one node to another by writing the process image directly to a remote node	replace stable storage with memory and processor redundancy	handle fault at a time and availability of one checkpoint in case the other is lost
Overhead .	large storage, very high log management and work in small process	minimum storage overhead and negligible overheads in failure free executions	high latency and memory and disk overhead	High memory overhead for storing checkpoints	low memory overhead

Advantages:

Failure free performance overhead is low compared to other checkpoint based recovery techniques. Recovery from failure is much simpler as compared to uncoordinated check pointing. This is because a consistent set of checkpoints need not be established; instead, a global consistent checkpoint is maintained in the system at any point of time. Coordinated check pointing is not susceptible to Domino effect since every process upon failure always restarts from the most recent checkpoint. Unlike asynchronous check pointing, the system does not maintain any useless checkpoints. Each process has to maintain only one permanent checkpoint on stable storage.

Thus the storage overhead is reduced. Moreover, there is no need for garbage collection. Different checkpoints

- **Uncoordinated CheckPointig:** Most convenient and save their checkpoints individually.
- **Coordinated Check pointing:** Coordinated checkpointig are not suffered from rollback propagations and processes save their state together.
- **Communication Induced Check pointing:** Communication Induced Check pointing is preventing domino effect, piggybacking and information of regular message exchanged by the processes.
- **Diskless Check pointing:** Improve performance in distributed / parallel applications and process migration save process image.
- **Double check pointing:** uses in small memory footprint on large number of processors. exscientific applications.

Disadvantages:

A process may take a useless checkpoint that will never be part of a global consistent state. Furthermore, each process maintains multiple checkpoints and has to periodically invoke a garbage collection algorithm to reclaim the checkpoints that are no longer useful. Besides,this method is not suitable for applications with frequent output commits because theserequire global coordination to compute the recovery line. Determining a consistent global checkpoint may involve lot of overhead, especially in large systems, and the processes may have to be restarted from the beginning due to the non-existence of a consistent global checkpoint other than the initial state.
Different checkpoints.

- **Uncoordinated CheckPointig:** Unsuitable, domino effect, wastage memory, unbounded & complex garbage collection.
- **Coordinated Check pointing:** Consistent checkpoint and large latency for saving the checkpoints storage.
- **Communication Induced Check pointing:** Deteriorated parallel performance & requires standby processors.
- **Diskless Check pointing:** Communication bottleneck.
- **Double check pointing:** depend on a central reliable storage and required additional hardware.

Conclusion:

It is reasonable to say that the major source of overhead in check pointing schemes is the stable storage latency. Communication overhead becomes a minor source of overhead as the latency of network communication decreases. In this scenario, the coordinated checkpoint becomes worthy since it requires less accesses to stable storage then uncoordinated checkpoints. Furthermore, in practice, the low overhead gain on uncoordinated check pointing do not justify neither the complexities of finding the recovery line after failure and performing the garbage collection nor the high demand for storage space caused by multiple checkpoints of each process. Check pointing protocols require the processes to take periodic checkpoints with varying degrees of coordination. At one end of the spectrum, coordinated check pointing requires the processes to coordinate their checkpoints to form global consistent system states. Coordinated check pointing generally simplifies recovery and garbage collection, and yields good performance in practice. At the other end of the spectrum, uncoordinated check pointing 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 check pointing 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 nondeterministic nature of these protocols complicates garbage collection and degrades performance.

References:

[1] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: an approach to designing fault-tolerant distributed computing systems", ACM Transactions on Computer Systems, 1 **(1985)**.

[2] H. F. Li, Z. Wei and D. Goswami, "Quasi-atomic recovery for distributed agents", Parallel Computing, 32 **(2006)**.

[3] Y. Luo and D. Manivannan, "FINE: A Fully Informed aNd Efficient communication-induced check pointing protocol for distributed systems", J. Parallel Distrib. Comput., 69 **(2009)**.

[4] J. T. Rough and A. M. Goscinski, "The development of an efficient check pointing facility exploiting operating systems services of the GENESIS cluster operating system", Future Generation Computer Systems, 20, 4 **(2004)**.

[5] Bhargava, B. and Shu-Renn, L. ,"Independent Check pointing and Concurrent rollback for recovery in distributed Systems-an optimistic approach",n proceedings of The 17th Symposium on Reliable Distributed Systems, pp. 3-12. Columbus, USA, October 1988.

[6] Partha Sarathi Mandel, Krishnendu Mukhopadhaya, " Performance analysis of different check pointing and recovery schemes using stochastic model" Journal of Parallel and Distributed Computing , 66(1), pp. 99-107, January 2006

[7] Y.Manable. "A Distributed Consistent Global Checkpoint Algorithm with minimum number of Checkpoints", Technical Report of IEICE, COMP97-6 April, 1997

[8] S.Monnet, C.Morin, R.Badrinath, "Hybrid check pointing for Parllel Applications in Cluster Federations", In 4th IEEE/ ACM International Symposium on Cluster Computing and the Grid, Chicago, IL, USA, pp. 773-782, April 2004

[9] P. **A.** Lee and T. Anderson, *Fault Tolerance: Principles and Practice.* Springer-Verlag/Wien, 1990.

[10] A. Duda.(1983): The effects of check pointing on program execution time. Information Processing Letters, 16, pp. 221-229.