International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-6 Issue-3, February 2017

A Study on Check Pointing Protocols for Mobile Distributed Systems

Komati Sathish

Abstract: ***A large number of distributed checkpointing protocols have appeared in the literature.a distributed checkpointing protocol could be the best in a specific environment, but not in another environment.Distributed snapshots are an important building block for distributed systems and are useful for constructing checkpointing protocols among other users. Communication-Induced Checkpointing protocols are classified into two categories in the literature:*** **Index-*****based and*** **Model-*****based.Recently, more attention has been paid to providing checkpointing protocols for mobile systems. check point 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. This paper surveys the protocols which have been appeared in the literature for checkpointing in mobile distributed systems.***

Keywords: ***Checkpoint/restart, checkpointing protocols, Distributed systems, rollback recovery, fault tolerant computing.***

I. INTRODUCTION

A distributed system containing more several processes that execute on geographically dispersed computers and collaborate via message-passing with each other to achieve a common goal[1].checkpoint is one of the most prominent techniques for providing fault-tolerance, and can also be used for debugginng and migration in both uniprocessor and distributed systems[2,3].particularly, checkpointing is the act of saving a program's state on stable storage, and restart is the act of restarting an application from its saved state. Especially, if an application takes periodic checkpoints ,then in case of failure, it is possible to restart iit from the latest checkpoint, thereby avoiding osing all the computation that was carried before that checkpoint.

Many distributed checkpointing protocols produce control overhead [4].control overhead is the over head due to control information. During the past years a large number of checkointing protocols have been proposed for distributed systems[5].Most of these protocols were never implemented or tested. The distributed mobile systems use checkpointing for providing fault tolerance. In this case, when fault or failures of process occur, an application with mobile should rollback to a consistent global checkpoint as close as possible to the end of the computation

Manuscript published on 28 February 2017.
* Correspondence Author (s)
Komati Sathish, Research Scholar, Department of Computer Science and Engineering, Sri Satya Sai University of Technology & Medical Sciences, Sehore, Bhopal (M.P). India. E-mail: komatisathish459@gmail.com

A local checkpoint is a recorded state of process. A global checkpoint is a set of local checkpoints one from each process in a distributed system [6].A consistent global checkpoint is one in which every message that has been received is also shown to have been sent in the corresponding state of sender.

A distributed mobile system contains of both static Mobile service Stations and Mobile Hosts.A set of wireless communication links and dynamic links can be established between an mobile service station and mobile host,and a set of high-speed communication link is assumed between the mobile service stations.An mobile service station may communicate with a number of mobile hosts but an mobile host communicates with the rest of the system via the mobile service station it is connected to.

II. CHECKPOINTING PROTOCOLS

Checkpointing is a standard method for the repair of faults in systems. The idea is to save the state of the system on a stable periodic to prevent breakdowns. That way when you restart after a power failure, the state saved newest restored and execution resumes its course before the crash. The overall status of a distributed system is defined by the union of local states of all processes belonging to the system. Taking checkpoints is the process of periodically saving the state of a running process to durable storage. Checkpointing allows a process that fails to be restarted from the point its state was last saved, or its checkpoint. If the host processor has not failed, temporal redundancy can be used to roll back and restart the process on the same platform. As in other systems, this method is widely used in grids [3,4]. Otherwise, if the host has failed, the process may be migrated, or transferred, to a different execution environment where it can be restarted from a checkpoint (a technique also referred to as failover). This section begins by discussing checkpoint and process migration methods used in commercial and science grid systems that are based on methods used in high performance cluster computing. This is followed by discussion of new methods being developed or adapted for scaled grid environments, together with related issues that need to be resolved. Most notable is the issue of finding efficient methods for checkpointing many concurrent, intercommunicating processes, so that in the event of failure, they can resume from a common saved state [9]. Checkpointing can be initiated either from within grid systems or within applications.

There are two main classes of protocols: coordinated checkpointing and message logging.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

A. *Coordinated Checkpointing Protocols*

Coordinated checkpointing is an attractive approach for transparently adding fault tolerance to distributed applications without requiring additional programmer efforts. In this approach, the state of each process in the system is periodically saved on stable storage, which is called a checkpoint of the process. To recover from a failure, the system restarts its execution from a previous error free, consistent global state recorded by the checkpoints of all processes. More specifically, the failed processes are restarted on any available machine and their address spaces are restored from their latest checkpoints on stable storage. Other processes may have to rollback to their checkpoints on stable storage in order to restore the entire system to a consistent state. Coordinated checkpointing simplifies failure recovery and eliminates domino effects in case of failures by preserving a consistent global checkpoint on stable storage. However, the approach suffers from high overhead associated with the checkpointing process. Two approaches are used to reduce the overhead: First is to minimize the number of synchronization messages and the number of checkpoints, the second is to make the checkpointing process non blocking. The protocol requires processes coordinate their checkpoints to form a consistent global state. A global state is consistent if it does not include any orphan messages (i.e, a message received but not already sent). This approach simplifies the recovery and avoids the domino effect, since every process always restarts at the resume point later. Also, the protocol requires each process to maintain only one permanent checkpoint in stable storage, reducing the overhead due to storage and release of checkpoints (garbage collection) Its main drawback however is the large latency that require interaction with the outside world, in this case the solution is to perform a checkpoint after every input / output. To improve the performance of the backup coordinated, several techniques have been proposed. We have implemented as non-blocking coordinated checkpointing and Communication induced checkpointing

1) Non-Blocking Coordinated Checkpointing: 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 dealt with, it may result in an inconsistency. This algorithm uses markers to coordinate the backup, and operates under the assumption of FIFO channels. a comparison of protocols for coordinated checkpoint blocking and non-blocking has been made. Experiments have shown that the synchronization between nodes induced by the protocol blocking further penalize the performance of the calculation with a non-blocking protocol. However, using frequencies of taken checkpoints usual performance of the blocking approach is better on a cluster to high-performance communications.

2) Communication induced checkpointing: This protocol defines two types of checkpoints [1]: local checkpoints taken by processes independently, to avoid the synchronization of coordinated backup and forced checkpoints based on messages sent and received and dependency information carried 'piggyback' on these posts, so to avoid the domino effect of uncoordinated backup, ensuring the advancement of online collection. Unlike coordinated checkpoint protocols, the additional cost due to the medium access protocol disappears because the protocol does not require any message exchange to force a checkpoint: this information is inserted piggyback on the messages exchanged.

B. *Message-Logging Protocols*

Message logging is a common technique used to build systems that can tolerate process crash failure. These protocols required that each process occurs. Indeed, during the process execution, the determinants of messages are stored in volatile memory, before being saved periodically on stable support. The storage stable memory is asynchronous: the protocol does not require the application to be blocked during the backup memory stable. Induced latency is then very low. However, a failure may occur before the messages are saved on stable storage. In this case, the information stored in volatile memory of the process down is lost and the messages sent by this process are orphaned. This can produce a domino effect of rollbacks, which increases the recovery time.

Thus, message logging protocols implement an abstraction of a resilient process in which the crash of a process is translated into intermittent unavailability of that process. All message logging protocols require that the state of a recovered process be consistent with the states of the other Processes. This consistency requirement is usually expressed in terms of orphan processes, which are surviving processes whose states are inconsistent with the recovered state of crashed process. Thus, in the terminology of message logging, message logging protocols must guarantee that there are no orphan processes, either through careful logging of through a somewhat complex recovery protocol. The logging mechanism uses the fact that a process can be modeled as a sequence of deterministic state intervals, each event begins with a non-deterministic. An event may be receiving a message, or issued or other event in the process. It is deterministic if from a given initial state, it always happens at the same final state. [1] The principle of Logging is to record on a reliable storage any occurrences of non-deterministic events to be able to replay them in recovering from a failure. During execution, each process performs periodic backups of their states, and recorded in a log information about messages exchanged between processes.

There are three message-logging categories: pessimistic, optimistic, and causal.

i) Pessimistic Message-Logging

This protocol was designed under the assumption that a failure may occur after any nondeterministic event (i.e. message reception). Then, each message is saved on a stable storage before to be delivering to the application.

These protocols are often made reference to the synchronized because when logging process logs an event of nondeterministic stable memory, it waits for an acknowledgment to continue its execution.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

In a pessimistic logging system, the status of each process can be recovered independently. This property has four advantages:

- Process can send messages to the outside without using a special protocol
- The process restarted at the most recent checkpoint.
- Recovery is simple because the effects of a failure are limited only on the fail process
- The garbage collector is simple

The main drawback is the high latency of communications, which results in degradation of the applications response time. Several approaches have been developed to minimize synchronizations:

- The use of semiconductor memories such as non-volatile stable support
- The sender based message logging (SBML) [14] which preserves the determinant or the message in the volatile memory of the transmitter, instead of a remote memory

ii) Optimistic Message-Logging

This protocol uses the assumption that the logging of a message on reliable support will be complete before a failure, the determinants of messages are stored in volatile memory, before being saved periodically on stable support. The storage stable memory is asynchronous: the protocol does not require the application to be blocked during the backup memory stable. Induced latency is then very low. However, a failure may occur before the messages are saved on stable storage. In this case, the information stored in volatile memory of the process down is lost and the messages sent by this process are orphaned. This can produce a domino effect of rollbacks, which increases the recovery time.

iii) Causal Message-Logging

This protocol combines the advantages of both previous methods. As optimistic logging, it avoids the synchronized access to stable, except during the input / output. As pessimistic logging, it allows the process to make interactions with the outside world independently, and does not create process orphan. Causal logging protocols piggyback determinants of messages previously received on outgoing messages so that they are stored by their receivers.

III. CHECKPOINTING PROTOCOLS IN COMPARISON

Many checkpointing protocols were incepted at a time where the communication overhead far exceeded the overhead of accessing stable storage. Furthermore, the memory available to run processes tended to be small. These tradeoffs naturally favored uncoordinated checkpointing schemes over coordinated checkpointing schemes. Current technological trends however have reversed this tradeoff. In modern systems, the overhead of coordinating checkpoints is negligible compared to the overhead of saving the states [10]. Using concurrent and incremental checkpointing, the overhead of either coordinated or uncoordinated checkpointing is essentially the same. Therefore, uncoordinated checkpointing is not likely to be an attractive technique in practice given the negligible performance gains. These gains do not justify the complexities of finding a consistent recovery line after the failure, the susceptibility to the domino effect, the high storage overhead of saving multiple checkpoints of each process, and the overhead of garbage collection. It follows That coordinated checkpointing is superior to uncoordinated checkpointing when all aspects are considered on the balance. A recent study has also shed some light on the behavior of communication-induced checkpointing [2]. It presents an analysis of these protocols based on a prototype implementation and validated simulations, showing that communication-induced checkpointing does not scale well as the number of processes increases. The occurrence of forced checkpoints at random points within the execution due to communication messages makes it very difficult to 22 predict the required amount of stable storage for a particular application run. Also, this unpredictability affects the policy for placing local checkpoints and makes CIC protocols cumbersome to use in practice. Furthermore, the study shows that the benefit of autonomy in allowing processes to take local checkpoints at their convenience does not seem to hold. In all experiments, a process takes at least twice as many forced checkpoints as local, autonomous ones.

Table-1 Comparison Between Different Checkpoits Protocols

	1	2	3	4	5
A	complex	simple	simple	complex	complex
B	several	1	1	several	1
C	possible	no	no	no	no
D	Un bounded	Last global check point	Last global check point	Possibley several checkpoints	Last checkpoints
E	yes	no	no	yes	yes
F	Not possible	Very slow	fastest	slow	fast

1. Uncoordinated checkpointing. 2. Coordinated checkpointing. 3. Pessimistic Logging. 4. Optimistic Logging. 4. Causal Logging

A. Garbage collection. B. Checkpoints per process

C. Domino effect. D. Orphan processes E. Rollback extent

IV. RELATED WORK

There has been much work on checkpointing performance analysis [11, 12, 15, 17]. Most of these works do not take into account the rollback propagation. Ours is the first to incorporate all parameters that affect the performance in distributed environments into an analytical measure. Mishra and Wang [11] evaluated several checkpointing protocols by implementing and running them with test applications. Ziv and Bruck [17] compared four checkpointing protocols by using the Markov Reward Model [13]. Our approach differs from [17] in that we provide a technique for comparing any checkpointing protocol based on rollback propagation. ziv and Bruck presented in [18] a checkpoint scheme for duplex systems, and conducted a performance analysis for their scheme in the duplex system.

However, it is not a general system for distributed executions. Vaidya defined the overhead ratio for uniprocessor systems as a function of the checkpoint overhead and latency [14], and proved that the optimum checkpoint interval depends on o. Additionally, he claimed that the overhead ratio can be computed for distributed systems as in uniprocessor systems by taking the values of parameters either to be the maximum or the average over all processes. In [16], Vaidya computed the overhead ratio for the two-level recovery approach. This approach tolerates single failures with a low overhead and multiple failures with a higher overhead. Plank and Thomason [14] presented a method for estimating the overhead ratio for coordinated checkpointing. By assuming coordinated checkpointing, they do not care about rollback propagation. Moreover, they do not address the control overhead incurred by control information.

V. CONCLUSION &RESULT

we have reviewed some fundamental concepts of checkpointing protocols in distributed systems. This paper presents a comprehensive model of rollback recovery protocols that encompasses a wide range of check point/restart protocols. included coordinated checkpoint and uncoordinated checkpoint protocols. This model provides the first tool for a quantitative assessment of all these protocols. Hence the concept of checkpoint is introduced before planned disconnection so that checkpointing can be completed without any delay resulting enhanced fault tolerance in the proposed scheme.

REFERENCES

1. Ch.D.V.Subba Rao and MM Naidu : A new efficient coordinated checkpointing protocol combined with selective sender based message logging , IEEE,2008.
2. Acharya and B.R.Badrinath ,checkpointing distributed Applications on Mobil computers,proc.3rd Int'l conf.parallel and distributed Information systems, sept.1994.
3. R.Prakash and M.Singhal, "Low-cost checkpointing and failure recovery in mobile computing systems," IEEE Trans.parallel and distributed systems pp.1035-1048,oct 1996
4. Lalit kumar p.kumar "A synchronous checkpointing protocol for mobile distributed systems: probabilistic approach" Int.Journal of information and computer society 2007.
5. G.H.Forman and J.Zahorjan, The changes of Mobile computing ,computer pp 38-47,Apr-1994
6. Ms.Pooja Sharma and Dr.Ajay khuntala " A survey of checkpointing Algorithm in Mobile Ad Hoc Network"Globl Journal of Computer Science and Technolgy 2012.
7. Sarmistha Neogy,Anupam siha,pradip k Das ,CCMUL: A Checkpoinying protocol for distributed system processes,IEEE,2004.
8. B.bhargava,S.R.Lian "Independent checkpointing and concurrent rollback for recovery in distributed systems-An Optimistic approach". proc 7th IEEE Symp.Rliable Distributed syst. pp 3-12 1988 oct.
9. L. Alvisi, E.N. Elnozahy, S. Rao, S. A. Husain and A. Del Mel. “An analysis of communication-induced checkpointing.” In Proceedings of the Twenty Ninth International Symposium on Fault-Tolerant Computing, Jun. 1999.
10. D.B. Johnson. “Distributed system fault tolerance using message logging and checkpointing.” Rice University, Dec. 1989.
11. S. Mishra and D. Wang. Choosing an Appropriate Checkpointing and Rollback Recovery Algorithm for LongRunning Parallel and Distributed Applications. In 11th ISCA International Conference on Computers and their Applications, San Francisco, CA, March 1996
12. J. S. Plank and M. G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. Journal of Parallel and Distributed Computing, 61(11):1570–1590, November 2001.
13. K. S. Trivedi. Probability and Statistics with Reliablity, Queuing, and Computer Scince Applications. Prentice-Hall, USA, 1982.
14. Nitin Vaidya. On Checkpoint Latency. In Pacific Rim International Symposium on Fault-Tolerant Systems, Newport Beach, December 1995.
15. Nitin H. Vaidya. Another Two-Level Failure Recovery Scheme: Performance Impact of Checkpoint Placement and Checkpoint Latency. Technical Report TR94-068, Deprt. of Computer Science, Texas A&M University, 1994.
16. Y. M. Wang. Consistent Global Checkpoints that Contain a Given Set ofCheckpoints. IEEETransactions on Computers, 42(4):456–486, April 1997.
17. Ziv and J. Bruck. Analysis of Checkpointing Schemes for Multiprocessor Systems. In Proceeding ofthe 13th Symposium on Reliable Distributed Systems, pages 52–61, 1994.
18. Ziv and J. Bruck. Efficient checkpointing over local area network. In IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, June 1994.

ABOUT AUTHOR

Komati Sathish is a Research Scholar in the Department of Computer Science and Engineering (CSE) at Sri Satya Sai University of Technology &Medical Sciences, Sehore, Bhopal (Madhya Pradesh). India.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)