International Journal of Electronics Communication and Computer Engineering
Volume 3, Issue 1, ISSN 2249 –071X

An Overview of Checkpointing Techniques for Fault Tolerance in Distributed Computing Systems

Jagdish Makhijani, Dr. Anil Rajput
j_makhijani@yahoo.com

***Abstract*- Checkpointing is an important feature in distributed computing systems. It gives fault tolerance without requiring additional efforts from the programmer[1]. In order to provide fault tolerance for distributed systems, the checkpointing technique has widely been used and many researchers have been performed to reduce the overhead of checkpointing coordination. A *checkpoint* is a snapshot of the current state of a process. It saves enough information in non-volatile stable storage such that, if the contents of the volatile storage are lost due to process failure, one can reconstruct the process state from the information saved in the non-volatile stable storage [1].**

***Keywords*—Backup passive module, Transient, Orphan message, Domino effect, Coordinated check pointing, Livelocks problems**

I. INTRODUCTION

A distributed computing system is a collection of multiple autonomous computers that communicates through a computer network to achieve a common goal. These connected computers share the resources of the complete system in such a manner that users perceive the system as a single, integrated computing facility. The distributed computing system has several autonomous computational computers with their own local memory. These computers communicate with each other by message passing.

A fault is an undesired change in the internal structure or external environment of distributed computing system. This may occur at once or many times during the complete execution of a distributed program. The Fault Tolerance is the technique to complete the execution of distributed program successfully in the presence of faults. Checkpointing is a fault tolerance technique, generally used in distributed computing systems. The Checkpointing provides fault tolerance without requiring additional efforts from the programmer [1].

II. RECOVERY IN DISTRIBUTED COMPUTING SYSTEMS

The recovery is the process of recovering the last stable state of distributed computing system in the presence of faults. The recovery may be one of the following two types: Rollback Recovery and Roll Forward Recovery. The Rollback recovery and Roll Forward Recovery are generally called Backward Recovery and Forward Recovery respectively.

A. Rollback Recovery (Backward Recovery)

If the nature of faults is not certain, then the Rollback recovery (Backward Recovery) applies to the system. The rollback recovery may be used in the cases where the foreseeing the nature of the faults is not possible. The concept of Rollback recovery aims towards the storage of system states regularly to restore the system to a previous fault-free state when a failure puts system in an inconsistent state. The Rollback Recovery provides a concept of a general recovery mechanism to distributed system.

The rollback recovery restores a process to a prior consistent state. The consistent state can be achieved by saving the state of process after a specific time interval at the time of execution. These saved states can be used to restore the process to a prior state. These saved states are known as checkpoints.

The process of creating checkpoints may be applied on every active module or process. This process of saving checkpoint may be done in two ways for an active module or process [3].

- Each checkpoint is to be multicast to all its backup passive modules.
- Each checkpoint is stored on the local stable storage of process/module.

Checkpoints are normally stored in local and stable storage. The reason behind it is the guaranteed stability or permanence of the information stored. Because of this, the chance of losing information in Stable storage becomes almost negligible.

B) Roll-forward recovery

The roll-forward recovery technique is an example of the semi-active replication. In semi-active replication, a failure is detected by a TMR (triple-modular redundancy) voting or by a comparison mismatch between two active modules. The TMR voting is performed among three active modules by triggering a validation step. In the meanwhile the TMR voting is performed; all the participating processors continue execution and spare processor is used to determine which of the divergent processors is correct.

The other technique used for roll forward recovery is the look-ahead execution with rollback. In look ahead execution, the copies of a process are executed at different processors initially. The results of these executed copies of processes are compared or voted at the time of checkpoint. If the process of voting or comparison completed

successfully, a correct result may be obtained. This correct result is further saved in stable storage. The execution of these copies of the next task is done on the basis of these results.

In the case of failed voting, the execution of copies of next task is performed on the basis of each of the results of the previous task. Simultaneously a rollback execution of the previous task is implemented, i.e., the re-execution of the previous task is started on spare processors in order to obtain a correct result. Then the results of the next task are stored which are based on the correct versions of the previous task. The other results (which may be incorrect) are discarded. This method reduced the time consumed in rollback execution.

If all the versions fail (all the results are incorrect) or the re-execution of the previous task does not get a correct result, the rollback execution cannot be avoided.

In roll-forward recovery, the rollback time is saved by taking advantage of existing correct results without restarting from the beginning.

III. Basic Concept of Checkpointing

Checkpointing is a technique for inserting fault tolerance into distributed computing systems. It is used to tolerate the transient faults. It is one of the most important fault-tolerant services. It is used to restore the system to a consistent state after failure. The checkpointing technique basically stores a snapshot of the current state of process and later on uses it for restarting the execution of process in case of failure.

Checkpoints are created periodically, after specific time interval, during the normal execution of a active module or process. This information on every checkpoint is saved on a stable storage so that it can be used whenever system fails due to any fault. The information in a checkpoint contains the process state, its environment, the value of registers, etc. The process is rollback to the last saved state whenever any fault occurs. In a coordinated distributed computing system, the multiple processes communicate with each other, the matter becomes complicated. In this case, the system state includes the state of all the active modules or processes. Each active module or process can establish a checkpoint at any given time, but there is no way to establish a checkpoint of all the active modules or processes simultaneously [2].

Checkpointing and rollback-recovery are well-known techniques that allow processes to make progress in spite of failures due to presence of transient faults [4]. The failures under consideration are transient faults which are unlikely to recur when a process restarts. With this scheme, a process saves its state on stable storage from time to time to establish checkpoints [5]. When a failure occurs, the process rolls back to its most recent checkpoint, assumes the state saved in that checkpoint, and resumes execution.

IV. Properties of Checkpointing Technique

There are many different ways for achieving application checkpointing. Depending on the specific implementation, a tool can be classified as having several properties:

1) Amount of state saved: This property refers to the amount of information saved for a specific state of process. Instead of storing all information, selected necessary and specific relevant cores of data can be saved to restore a process in previous consistent state. The complexity of checkpoint can be decreased by saving the required information efficiently.
2) Automatization level: Depending on the effort needed to achieve fault tolerance through the use of a specific checkpointing solution.
3) Portability: Whether or not the saved state can be used on different machines to restart the application.
4) System architecture: How is the checkpointing technique implemented: inside a library, by the compiler or at operating system level.

V. Local and Global Checkpoints

A local checkpoint is the saved state of an active process or module after a regular time interval. The local checkpoint is saved on stable storage of active module or process. The local checkpoint is used to restart the execution of module or process in case of failure due to any transient fault.

A global checkpoint consists of a set of local checkpoints and can be used to restart the execution of a distributed computation program once the failure occurs.

Informally, a global checkpoint is called consistent if no local checkpoint in that set happens before [6] another one [7], [8].

VI. Consistent State of System

A system state is said to be consistent if it does not contain any orphan message. A message is said to be orphan if it is received by destination process, but the sender process's information is missing [9], [10]. In this case, receiver cannot find the sender of the message.

In order to record a consistent global checkpoint on stable storage, processes must synchronize their checkpointing activities. When a process takes a checkpoint, it asks (by sending checkpoint requests to) all relevant processes to take checkpoints. Another strategy of creating a consistent state is the logging of all messages by both sender and receiver.

VII. Chcekpointing in Distributed Computing System with Shared Memory

In Distributed Computing System with shared memory, checkpointing is a technique that helps long-running applications in tolerating the errors which leads to lose the effect of work. The main property which should be induced by checkpointing techniques in such systems is preserving system consistency in case of failure due to presence of faults. There are two main strategies used for checkpointing in such distributed computing systems, first one is coordinated checkpointing, and the second is communication induced independent checkpointing. In the coordinated checkpointing, all cooperating processes work together to establish coherent checkpoint while in the communication induced independent checkpointing (which is also called dependency induced), processes takes local checkpoints independently.

The global consistency cannot be ensured by forcing processes to create checkpoint at a fixed and regular time intervals. The checkpoints created by different processes still may not form a consistent state even if we postulate the existence of global clock. The need for establishing a consistent state may force other process to roll back to their checkpoints, which in turn may cause other processes to roll back to even earlier checkpoints. In the most extreme case it may mean that the only consistent state found is the initial state. This one is called domino effect.

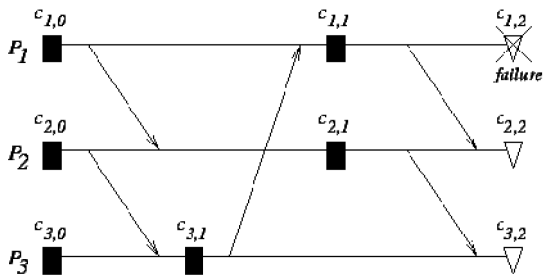

Fig. 1: Domino Effect

In the coordinated checkpointing approach, it should be ensured that the checkpoints are consistent. In communication induced checkpointing, each process checkpoints its own state independently.The system state may be saved either locally, in stable storage, or in a distant node's memory.

VIII.Independent and Consistent Checkpointing

The checkpointing schemes may also be categorized as following: independent checkpointing [10] and consistent checkpointing [11].

In independent checkpointing, there is no collaboration between processes on taking checkpoints and the set of consistent checkpoints to which processes roll back is determined upon the detection of a failure. These types of scheme suffer from the domino effect and livelocks problem. The Domino Effect is the phenomenon where the rollback of one process may result in the limitless cascade of rollbacks of other processes. In the livelocks problem a single failure can cause an infinite number of rollbacks, preventing the system from making progress.

The domino effect can be eliminated by message logging and replaying [l2-14] when the execution of processes is deterministic. However, if process execution is not deterministic (e.g., the computation accesses time-varying data or resources, or the computation depends on ordered messages), independent checkpointing cannot be effectively used. In the independent checkpointing, it becomes mandatory for a process to keep all the checkpoints that have been taken since program initialization to achieve fault tolerance.

In contrast to independent checkpointing, consistent checkpointing saves only two checkpoints for each process [15-16]. When a process fails, all the processes need only to roll back to their latest checkpoints (if necessary). The consistent checkpointing, the assumption on deterministic process execution is not required. This type of checkpointing, however, suffers from the disadvantage that they usually require a higher overhead in control messages, and usually has a two-phase-commit structure. The first phase is used to reach the related processes from the initiating (failed) process and to collect their status, and the second phase to deliver the checkpoint/rollback decision based on the collected status. In this scheme, less concurrency in process execution can be achieved if whenever a checkpoint (or rollback) is being attempted, all related application processes are suspended until the checkpoint (or rollback) is committed/aborted.

IX. Distributed Checkpointing

In distributed checkpointing the processes cooperate to establish their local checkpoint in such a manner that the set of checkpoints of all processes represents a consistent global state i.e., there are no orphan or lost messages. This checkpointing scheme avoids the domino effect. The disadvantage with this checkpointing is the complexity as it requires coordination among all processes. The advantage of this scheme is the simplified recovery. The other advantage is the low overhead since only a few checkpoints of each process needs to be saved.

X. PRACTICAL IMPLEMENTATION OF CHECKPOINT FACILITY

A. Usage

A checkpoint facility enables the intermediate state of a process to be saved to a file. Users can later resume execution of the process from the checkpoint file. This prevents the loss of data generated by long-running processes due to program or system failures, and it also facilitates debugging when the bug appears after the program has executed for a long time.

B. Design Goals

The goals to design a checkpoint facility are [18]:

1) Transparency: The implementation should not require availability of user source to run the checkpoint utility. User applications need only link to the checkpoint library, which will automatically change the startup routine and system call import table. Changing the startup routine allows user to inject optional checkpoint specific command line flags in the application and initialize checkpointing.

2) Correctness: Process execution gives the same results whether or not checkpoints are taken at runtime. Resuming a process from a checkpoint provides the same result as the original execution.

3) Minimal Performance Impact: The checkpoint facility writes the various memory segments of the application to a checkpoint file. Elapsed time is therefore directly proportional to process size [17].

4) Portability: The checkpoint tool should be portable, i.e., it should be run on various systems such as Windows/NT, AIX and FreeBSD UNIX systems. For this, a similar user level methodology on all OS should be implemented, which may prove to be easily portable. This may lead to be some code differences over different OS implementations.

5) Multiple Thread Support: Since the checkpoint facility will be use on Distributed Computing Systems, it should supports checkpointing of multithreaded applications.

C. The Block Diagram of Checkpoint Facility

The figure 2 shows the block diagram of checkpoint tool to be developed in any operating system:

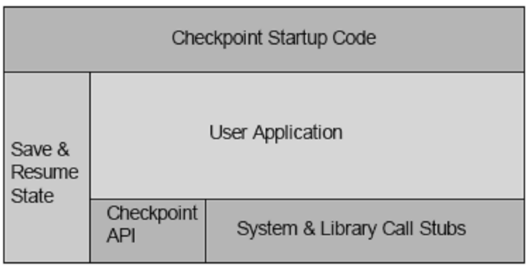

Fig 2: Block Diagram of Checkpoint Facility

D. A Basic algorithm for Checkpoint Facility

The basic algorithm for checkpointing in distributed computing system will be divided in two parts. The first part will be used to create checkpoints and the second part will be used to restore the system to previous consistent state in case of failure due to transient failure. Both part must be implemented on each active module or process to avoid the failure of system.

i. start
//PART-I: CREATION OF CHECKPOINT//
ii. Process the *Checkpoint Arguments*
iii. if *checkpoint* is required then
iv. create and save *checkpoint*
v. if *periodic checkpoint* is not saved then
vi. create and save *periodic checkpoint*
vii. if *checkpoint signal* received then
viii. create and save *checkpoint* for received signal
ix. else if *resume* is required then
x. create *dumpfile* for *resume checkpoint*
//END OF CHECKPOINT CREATION//

//PART-II: RESTORE TO CHECKPOINT//
xi. if *failure/error* occurs then
xii. {
xiii. restore *process* to previous *checkpoint*
xiv. if *system* is ok then
xv. *resume* to *execution*
xvi. else go to step *xiv*
xvii. }
xviii. end

XI. CONCLUSION

The checkpointing is a very important technique for Rollback Recovery. The Rollback Recovery is very effective in fault tolerance in Independent as well as Distributed Systems. The checkpointing should be done in very effective manner. For various conditions and systems, the method used for checkpointing may be different, but the result of those should be as much as effective.

REFERENCES

[1] Partha Sarathi Mandal, “Checkpointing and Self-Stabilization for Fault-Tolerance in Distributed Systems”, Ph.D. Thesis, 2006
[2] Vikas Agarwal, IIT Kanpur, “Fault Tolerance in Distributed Systems”
[3] www.cmpe.boun.edu.tr/courses/cmpe516/spring2007/Ertugrul%20Kuzan.ppt
[4] B. Randell, P. A. Lee, and P. C. Treleaven, "Reliability issues in computing system design," ACM Comput. Surveys, vol. 10, no. 2, pp. 123-166, June 1978.
[5] B. Lampson and H. Sturgis, "Crash recovery in a distributed storage system," Xerox Palo Alto Research Center, Tech. Rep., Apr. 1979.

[6] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System," Comm. ACM, vol. 21, no. 7, pp. 558-565, 1978.
[7] K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Trans. Computer Systems, vol. 3, no. 1, pp. 63-75, 1985.
[8] D. Manivannan, R.H.B. Netzer, and M.Singhal, "Finding Consistent Global Checkpoints in a Distributed Computation," IEEE Trans. Parallel and Distributed Systems, vol. 8, no. 6, pp. 623-627, June 1997.
[9] R.E. Strom and S.A. Yemini, "Optimistic Recovery in Distributed Systems," ACM Trans. Computer Systems, pp. 204–226, Aug. 1985.
[10] BHARGAVA, B., and LIAN, S.R.: 'Independent checkpointing and concurrent rollback for recovery in distributed systems – an optimistic approach'. IEEE Proc. symposium on Reliahk disrrihuted systems, 1988, pp. 3-12
[11] KIM, J.L., and PARK, T.: 'An efficient protocol for checkpointing recovery in distributed systems', ZEEE Trans. Parullrl and Distrib. Syst., Aug. 1993, 4, pp. 955-960
[12] JOHNSON, D.B., and ZWAENEPOEL, W.: 'Sender-based message logging'. Proceedings of 17th IEEE symposium on Fault-tolerant computing, June 1987, pp. 14-19
[13] JOHNSON, D.B., and ZWAENEPOEL, W.: 'Recovery in distributed systems using optimistic message logging and checkpointing'. Proceedings of 7th ACM symposium on Principles of distributed computing, Aug. 1988, pp. 171-181
[14] SISTLA, A.P., and WELCH, J.L.: 'Efficient distributed recovery using message logging'. Proceedings of 8th ACM symposium on Principles of' distributed computing, 1989, pp. 223-238
[15] KOO, R., and TOUEG, S.: 'Checkpointing and rollback-recovery for distributed systems', ZEEE Trans. Softw. Eng., Jan. 1987, 13, pp. 23-31
[16] LEU, P.Y., and BHARGAVA, B.: 'Concurrent robust checkpointing and recovery in distributed systems'. Proceedings of 4th IEEE international conference on Data engineering, 1988, pp. 154-163
[17] Johny Srouji, Paul Schuster, Maury Bach, Yulik Kuzmin: "A Transparent Checkpoint Facility on NT", Proceedings of the 2nd USENIX Windows NT Symposium Seattle, Washington, August 3–4, 1998
[18] Jie Wu: "Distributed System Design"

AUTHOR'S PROFILE

Jagdish Makhijani

He is working in Department of Information Technology, Rustamji Institute of Technology, BSF Academy, Tekanpur. He is M.Sc. in Computer Science from Jiwaji University, Gwalior and pursuing Ph.D. in Fault Tolerance in Distributed Computing Systems from Barkatullah University, Bhopal under the guidance of Dr. Anil Rajput.