ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 4, Issue 12, December 2016

A Study on VLG Recovery Scheme for Distributed Systems

Anand Jha, Sanjay Patsariya , Jankisharan Pahareeya, Aradhana Saxena

Assistant Professor, Department of Information Technology, Rustamji Institute of Technology,

BSF Academy, Tekanpur, Gwalior, Madhya Pradesh, India

ABSTRACT: Long running multiprocessor scientific applications are often subject to failures. A failure means loss of computation, sometimes significant. But all failures cannot be treated alike. Failures can be classified in many ways, for example, they can be classified into failures affecting one processor, multiple processors and system as a whole. Alternatively, they can be classified as transient and permanent failures. Another way to classify failures is as less-probable and high-probable failures. Irrespective of classification method, failures can be characterised by two aspects: impact intensity and occurrence frequency.
This paper introduces a three level coordinated checkpointing scheme, we call it VLG checkpointing scheme for multiprocessor and distributed systems. This three level VLG recovery scheme tolerates failures with less average performance overhead by considering above two aspects of failures.

KEYWORDS:failure, recovery, checkpointing, distributed, fault-tolerance.

I. INTRODUCTION

Mean time between failures (MTBF) of a multiprocessor or distributedsystem is considerably lower than a normal workstation, this means that a long running parallel or distributed application may encounter more failures during its execution. In absence of a failure recovery scheme, the application must be restarted from beginning whenever a failure occurs leading to unacceptable performance overhead. Thus, a fault tolerance provision is desirable that allow such applicationsto recover from failures to minimize loss of computation. Checkpointing is a method used bysuch applications to save their computational state at regular intervals on a stable storage, so that they may be restarted after interruptions without losing their computational progress [5]. However all such schemes are designed to tolerate arbitrary number of failures this means that all such recovery schemes treat all failures in the same way i.e. they treat entire system as faulty or fault-free, without consideration their characteristics except [3] where a two level recovery scheme is proposed.

Coordinated checkpointing algorithms are of two types: blocking and non-blocking [12,13, 14]. Blocking algorithms force all processes in the cluster computing system to block their computation during synchronization and checkpointing. In non-blocking algorithms, application processes are not blocked during synchronization. In non-blocking algorithms, although all processes are involved in creating the consistent global checkpointing state, processes are not firmly synchronized. In this scheme, in-transit and orphan messages may exist in the communication channel at the time the checkpoint is taken. However, processes employ message log to manage these messages and thus to guarantee that the checkpoint sets are consistent. Upon failure, the recovery of non-blocking coordination is more complicated than blocking coordination because not only the process needs to be restored, but also the communication channel should be recovered to the same state as the checkpoints were taken.

The Performance overhead of a recovery scheme is the increase in the execution time of the application when using a recovery scheme [2]. The performance overhead of a recovery scheme consists of two components:

1. Overhead during failure-free operation (failure-free overhead), e.g., checkpointing time and message logging.
2. Overhead during recovery (recovery overhead).

The design concept “make most frequent case fast” has been successfully used in designing many components of a computer system [6] and some aspects of fault tolerance [7, 8, 9, 10]. However, most distributed failure recovery schemes have ignored this advice. In any system, some failures have greater occurrence frequency as compared to other

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 4, Issue 12, December 2016

failures [2,4] causing partial (e.g. single processor) and total system failure.Most existing recovery schemes are identicalin the sense that they are designed to tolerate the worst case failure scenario i.e. they treat all failure alike, for example, the traditional implementations of consistent checkpointing algorithms are designed to tolerate total system failure [11].

But the design concept “most frequent case fast” suggests that a recovery scheme should provide lowoverhead protection against more probable failures, providing protection against other failures with possibly higher overhead.

An important parameter to reduce the coordinated checkpointing overhead is checkpointing frequency. If checkpoints are frequently taken, a larger overhead due to checkpointing will be incurred. Conversely, if checkpoints are taken too infrequently, a larger recovery overhead after system failures will be required.

Checkpoint frequency hasa direct effect on overhead if all failures are treated alike, it can be demonstrated [6] that the checkpoint frequency impacts more on the performance than the number of nodes involved in a checkpoint synchronization for both non-blocking and blocking schemes.

We propose a three-level approach to reduce the average performance overhead of failure recovery for long running applications by considering failure occurrence frequency and thus checkpoint frequency asthe critical parameter.

II. SYSTEM MODEL

Our computing model consists of N computing nodes as shown in Fig 1. Each computing node has a processor (so a processor or computing node are same), memory and a local disk. All the computing nodes in the system are identical. The computing nodes share a stable storage that can be accessed over the network. Each processor executes one process of a distributed application, therefore, a processor failure is synonymous with process failure. Computing nodes do not share a common memory or a common clock and message passing is the only way for nodes to communicate with each other. Messages are exchanged through reliable communication channels, whose transmission delays are finite but arbitrary.

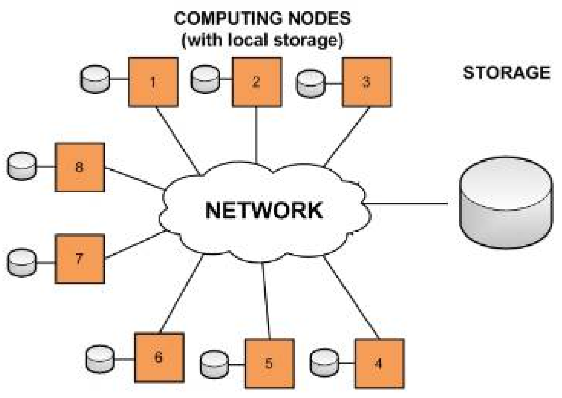

Figure 1

III. FAILURE MODEL

Failures in computer systems are usually classified into two types [14, 15,16]: (a) transient failure and (b) permanent failure. Transient failures include both soft errors (transient faults) in semiconductor devices [15,17,18] (e.g., memory or register bit errors, memory, and CPU transient bit-flip) and recoverable errors in other devices (e.g., disk read retries). Permanent failure or hardware failures [15, 19] are permanent physical defects whose repair normally requires component replacement, for example, a power supply or fan failure, permanent stuck-at faults in memory, permanent stuck-at faults in CPU registers, hard disk mechanical failures).

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 4, Issue 12, December 2016

The stable storage is assumed to be always failure-free. We only consider fail-stop failures of processors and local disks, Byzantine failures are not covered by our scheme. A processor is subject to transient as well as permanent failures. Transient failure of a processor results in the loss of its volatile memory contents; however, it does not cause a failure of its local disk. Thus, the checkpoints saved either in neighbours' volatile memory or its own local disk can be used to survive transient processor failure because in the case of a transient processor failure, the local disk of the faulty processor or neighbour's volatile memory becomes accessible once the processor comes back up after failure. However, the neighbour node of the failed node can also fail (processor or disk), in this case, it is not possible to recover from transient processor failure using memory based checkpoint but local disk and remote stable storage can be used. It can be noted that, in order to access the neighbour's volatile memory or its own local disk, the node's processor itself must be operational, so a permanent processor failure of a node cannot be recovered without using stable storage.

We assume that failure of a local disk always crashes the associated processor since local disk of a workstation often stores swapped out process memory, temporary files accessed by an application, as well as many files that are accessed by the operating system.

So we can say that a permanent failure (processor or disk) always crashes the associated computing node.

We assume failures of multiple processors are independent of each other; similarly, the local disk failures are independent.

IV. VLG CHECKPOINTING SCHEME

Since there are three levels of failures as per their occurrence frequency: Soft transient failures hasthe highest frequency of occurrence. Recoverable transient failures has medium frequency of occurrence. Permanent failurehas the least frequency of occurrence, we propose three types of checkpoints namely: V-Checkpoint, L-Checkpoint, and G-checkpoint.

V-checkpoint is created by a single processor (node) by saving its state in neighbour's volatile memory.The cost (failure-free checkpoint overhead) is C_V and checkpointing interval is T_V.

L-checkpoint is created by taking a checkpoint by all processors (nodes) simultaneously in a blocking coordinated mannerby saving its state in local disk. The cost for L-checkpoint is C_L and checkpointing interval is T_L.

G-checkpoint is created by taking a checkpoint by all processors (nodes) simultaneously in a blocking coordinated manner but this time not only saving it in local disk but also sending this checkpoint to remote stable storage. The cost for G-checkpoint is C_G and checkpointing interval is T_G.
We assume that processes take equidistant checkpoints i.e. every K^{th} checkpoint is L-checkpoint and every N^{th} checkpoint is G-checkpoint. So, $T_L = K \times T_V$ and $T_G = N \times T_V = (N/K)\, T_L$.
It can be said that $C_V<C_L<C_G$and $T_V<T_L<T_G$because of three reasons:

1. Blocking coordinated checkpointing, congest network (to access remote stable storage) besides accessing the same stable storage and
2. Network and memory bandwidths are faster than the bandwidth of storage systems (local disk and stable storage).
3. Soft transient failures has the highest frequency of occurrence, recoverable transient failures, has medium frequency of occurrence, permanent failure has the least frequency of occurrence

It is noted that L-checkpoint and G-checkpoint are consistent but V-checkpoint may not.

V. VLG RECOVERY SCHEME

It is obvious that V-checkpoint is used to recover from most frequent failures, L-checkpoint is used to recover from medium frequent failures and G-checkpoint is used to recover from least frequent failures. Possible cases of recovery using VLG scheme are discussed below.

I. Usage of V-Checkpoint:

Case A: Single processor transient failure

Single processor transient failure due to soft transient errorscan be recovered by rolling back faulty processor its last V-checkpoint. Since V-checkpoint is not consistent, the messages that the faulty process had received after the V-checkpoint and before failure,are resent in the appropriate order by the senders as these messages and order information are available in the volatile memory of message senders. In case the last V-checkpoint is not nearest checkpoint (an L-checkpoint is taken after V-checkpoint), then nearest L-checkpoint can also be used.

Case B: Multiprocessor transient failure

1. If the same processor fails again during recovery, then recovery is reinitiated.
2. If, however, another processor fails before faulty recovers from the first failure, then there are three possibilities:
   a. If the two faulty processors did not exchange any messages since their previous V-checkpoint (because senders need to resend messages), then the two simultaneous failures can be recovered by rolling back faulty processorsto their respective V-checkpoints.

   b. If a processor and its neighbour (where its V-checkpoint is stored) both failed simultaneously (simultaneous failure includes, a processor failed during the recovery of another processor), then V-checkpoint cannot be used for recovery.

   c. If two failed processor exchanged messages since their last V-checkpoints, then V-checkpoints cannot be used for recovery.

II. Usage of L-Checkpoint:

Case A: Single processor transient failure

Single processor failure because of soft and recoverable transient errors can be recovered by rolling back each processorto last L-checkpoint only when the nearest checkpoint is not V-checkpoint. Since L-checkpoint is consistent, the messages logging is not required.

Case B: Multiprocessor transient failure

1. If a processor and its neighbour (where its V-checkpoint is stored) both failed simultaneously (simultaneous failure includes, a processor failed during the recovery of another processor), then L-checkpoint can be used to recover by having each processor rollback to their last L-checkpoint.

2. L-checkpoint can be used to recover from all simultaneous failures even if the two faulty processors exchanged messages by rolling back each processor to the last L-checkpoint.

It is possible to use V-checkpoint with some encoding techniques to recover from simultaneous failures but such encoding techniques are highly time and memory consuming[6].

III. Usage of G-Checkpoint:

The G-checkpoint can be used to recover from all permanent failures (processor and disk) including catastrophic system failure. G-checkpoint can be used to recover from transient failure as well but V-checkpoint and L-checkpoint are having less cost compared to G-checkpoint for transient failures.

VI. CONCLUSION AND FUTURE WORK

This paper illustrates a three level VLG checkpointing scheme for recovery in multiprocessor and distributed systems by establishing a connection between checkpointing overhead and occurrence frequency of failures to minimize average performance overhead.

As per [2,3], for multiple transient failures, recovery is done by rolling back to G-checkpoint but our scheme never rollbacks to G-checkpoint for transient failures.

Although the VLG checkpointing scheme looks promising but a formal analysis needs to be done in order to determine the optimal value of T_V and to find K for T_L and N for T_G.

We considered equidistant checkpointing schedule. It would be motivating to examine VLG checkpointing scheme with an alternative checkpointing schedule, for example, let V-checkpoints need not be equidistant.

Our approach considers average performance overhead as metric. It would be interesting to evaluate the effectivenessof our scheme with respect to other metrics, for example,the probability of meeting a givendeadline.

Further, our VLG recovery scheme can be studied with various encoding schemes for V-checkpoint.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 4, Issue 12, December 2016

REFERENCES

1. Nitin H Vaidya, A Case for Two-Level Distributed Recovery Schemes, Department of Computer Science, Texas A & M University
2. Nitin H Vaidya, A Case for Two-Level Recovery Schemes, IEEE Trans. 1998
3. Nitin H Vaidya, A Case for Multi-Level Distributed Recovery Schemes, Department of Computer Science, Texas A & M University, May 1994
4. M. Lotfi , S. A. Motamedi, and M. Bandarabadi, Lightweight blocking coordinated checkpointing for cluster computer systems, in Proceedings of the 41st Southeastern Symposium on System Theory, 2009, pp. 144-147
5. L. M. Silva, J. G. Silva, Using two level storage for efficient checkpointing, IEEE proceedings online no. 19982440, 1998.
6. J.L. Hennessy and D.A. Patterson, Computer Architeture: A Competitive Approach, second ed. Morgan Kaufmann, 1996.
7. E. Gelenbe, “A Model for Roll-Back Recovery with Multiple Checkpoints,” Proc. Second Int’l Conf. Software Eng., pp. 251-255, Oct. 1976.
8. E. Gelenbe, “Model of Information Recovery Using the Method of Multiple Checkpointing,” Automation and Control, vol. 4, pp. 595-605, Apr. 1979.
9. J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
10. J.E.B. Moss, Nested Transactions: An Approach to Distributed Reliable Computing. Boston: MIT Press, 1985.
11. J.S. Plank, “Efficient Checkpointing on MIMD Architectures,” PhD thesis, Dept. of Computer Science, Princeton Univ., June 1993.
12. Tezzaron Semiconductor, “Soft errors in electronic memory - a white paper.”
13. Partha Sarathi Mandal and Krishnendu, Mukhopadhyaya, Concurrent checkpoint initiation and recovery algorithms on asynchronous ring networks, J. Parallel Distrib. Comput. 64 (2004) 649–661
14. Guohong Cao, Mukesh Singhal, Checkpointing with mutable checkpoints, Theoretical Computer Science 290 (2003) 1127 – 1148.
15. CHARNG-DA, “SCALABLE DISKLESS CHECKPOINTING FOR LARGE PARALLEL SYSTEMS”, Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2005
16. “DRAM soft error rate calculations,” Tech. Rep. TN-04-28, Micron Technology, Inc, 1994.
17. Actel Corporation, “Understanding soft and firm errors in semiconductor devices,” 2002.
18. T. Lin and D. Siewiorek, “Error log analysis: Statistical modeling and heuristic trend analysis,”
19. “Module mean time between failures (MTBF),” Tech. Rep. TN-04-45, Micron Technology, Inc, 1997.