International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 10, October- 2014

Tolerating Failure in Distributed Systems using Diskless Checkpointing

K. C. Maheswari [1], V. Hemalatha[2], P. B. Selvapriya[3]
Assistant Professor, Department of CSE,
N.S.N. College of Engineering and Technology, Karur,
Tamilnadu. India

Abstract :- **Checkpointing is a technique to perform fault tolerance in distributed computing systems. Diskless checkpointing stores checkpoint data in main memory instead of storing it in a secondary memory like disks. Diskless checkpointing is a technique to tolerate multiple failures in a distributed system using simple checkpointing and failure recovery, without depends on selected checkpoint processors. A neighbour based checkpoint approach stores checkpoint in checkpoint storage nodes which can tolerate only less number of failures in a single step and the checkpoint data in the main memory reduces the speed of the processor. To reduce the memory space required for saving checkpoints and increase the fault tolerance rate in distributed environment a LZW compression technique is used. This reduces the amount of checkpoint data to be stored in memory. LZW is a dictionary based compressor which replaces repetitive data patterns with compact representation. No need to pass the dictionary to the decompressor in LZW compression method. To tolerate more failures in the system the failure processes are distributed to different groups and recovered.**

Keywords - Distributed System; Diskless heckpointing; LZW Compression; Multiple Failures; Fault Tolerance

I. INTRODUCTION

Fault tolerance is one of the most desirable properties for many distributed or parallel computing systems. Previous fault tolerance methods involved in checkpointing the system state and restore it when a system is in failure. It is the method available to system engineers whose goal is to create of a robust, fault-tolerant system. A partial failure of a system may easily halt the operation of the entire system. So, many distributed systems use the checkpoint/rollback recovery technique [7] which allows a system to continue its process after an appropriate recovery action is taken after failure occurs. Normally, processors checkpoints must be stored in disk-based non-volatile storage. Although non-volatile storage can protect against hardware and power failures, the latency of writing checkpoints to a hard disk and reading from the disk incurs significant overhead for the system and may result in significant performance degradation.

There are various techniques [10] to minimize this source of overhead. These techniques include incremental checkpointing, memory exclusion and compression. However, the performance of non-volatile storage medium remains a major concern with all of these techniques for the distributed systems.

Diskless Checkpointing [4] [6] is a technique for storing the checkpoint of a long-running process on a distributed system without relying on non-volatile storage. It reduces the performance bottleneck of normal checkpointing on distributed systems. Diskless checkpointing makes checkpoint/rollback recovery is possible in distributed computing in which non-volatile storage is not readily available, for example in mobile computing systems the non-volatile storage will not readily available. A problem faced by diskless checkpointing is the increased usage of memory incurred by its implementation. So, a diskless checkpointing scheme should reduce the usage of memory as much as possible.

Diskless checkpointing methods fall into three categories: neighbour based [1] [3], parity-based, and Reed-Solomon coding-based [1]. In neighbour based diskless checkpointing, each processor saves its checkpoints in the memory of nearby processors. Each checkpoint is stored in its entirety in peer memory, and no coding is involved. Parity-based scheme use a predefined checkpoint processor to save the parity codes of the checkpoints taken by all the processors using XOR operations. The Reed-Solomon coding-based technique encodes checkpoints of multiple processors using Galois Field arithmetic. When a fault occurs in the system, a consistent checkpoint can be restored for each failed processor through the decoding process.

Existing parity-based and Reed-Solomon coding-based techniques usually require more pre-assigned processors for checkpointing the encoded data. These checkpoint processors are not participants of the original system, and their addition increases failure probability. In neighbour based system saving checkpoints in peer processors reduce the processor speed because of large amount of data in main memory. To reduce the size of the checkpoint data we use Lempel-Ziv-Welch (LZW) compression technique. This reduces the amount of checkpoint data to be stored in main memory and increases the processor speed. Then to increase the fault tolerance rate the failures are distributed among nodes in different groups.

II. RELATED WORK

Diskless checkpointing was proposed to avoid overhead associated with non-volatile storage. Traditional diskless checkpointing methods [9] were classified into the following categories: neighbor-based, parity-based, and Reed-Solomon code-based. In neighbour based diskless checkpointing, each processor stores its checkpoints in the memory of nearby processors. When a processor fails, the checkpoint data can be recovered from the corresponding checkpoint processor. The neighbor-based checkpointing scheme [5] has the following three categories: mirroring [2], pairing [2] and ring structured [1].

A. Neighbour Based Scheme:

In mirroring scheme [2], each processor in the distributed system is assigned a separate checkpoint processor in which it stores checkpoints. The pairing scheme organizes the distributed processors into pairs. Each processor sends checkpoints to its neighbor in the pair and, in return, receives and stores checkpoints for the neighbour processor. Therefore, separate checkpoint processors are not necessary. The ring-structured scheme combines all processors into a virtual ring. Each processor sends its checkpoints to the following nearby processor. The neighbor-based approach is simple. However, a failed processor cannot recover its state if its partner or neighbor storing its checkpoint fails at the same time.

B. Parity Based Scheme:

The parity-based diskless checkpointing technique [2] requires that all distributed processors have to take checkpoints, and the parity data for that checkpoint is saved in the main memory of its parity processor. When an application processor fails, the parity processor and all the other processors in that distributed system still working will cooperate to decode the last checkpoint for the failed processor. Therefore, the amount of checkpoint data to be stored is less in the parity-based technique. At the same time, the time overhead for checkpointing and failure recovery operations depends on the number of application processors in the system.

C. Reed-Solomon Code Based Scheme:

The Reed-Solomon code based diskless checkpointing technique works by encoding the checkpoints of n application processors to generate an extra set of k distinct checksum data. These k pieces of encoded information are stored at k dedicated checkpoint processors. When some application processors fail, the system is able to decode the original checkpoints for each of the failed application processors as long as the total number of failed processors (including both application processors and checkpoint processors) is no more than k. Reed-Solomon erasure code is used for encoding and decoding purposes.

III. PROPOSED SYSTEM

A. System Model

Consider a distributed system with n processors (P_0, P1, P_2,P_{n-1}) that are connected by a wired are wireless network. Assume that a task is divided into subtasks and are distributed among n processors. Subtasks communicate with each other by passing messages between them. Every processor in the system takes checkpoint according to some criteria. The checkpoint data is compressed using Lempel-Ziv-Welch compression technique. Then the compressed data is passed to the peer processes. The processes form a many groups. When failure occurs the failures are distributed among different groups and are recovered using the checkpoint data stored in the processes.

B. Operation Of The Proposed Model

The goal of the system is to tolerate more failures and to reduce the amount of data to be stored in main memory. To reduce the size of the checkpoint data that are passed to the peer processes are compressed using LZW compression technique. Every processor takes the checkpoint data in a particular time interval. Checkpoint data is compressed and transferred through the network to the peer processes. Compression also maximizes the effective bandwidth because less data is transferred. The principle of compression is to replace repetitive data patterns with compact representation. LZW compressor is used for compressing the checkpoint data.

LZW is referred to as a substitutional or dictionary-based encoding algorithm. The algorithm builds a data dictionary of data occurring in an uncompressed data stream. Patterns of data or substrings that are identified in the data stream and are matched to entries in the dictionary. If the substring or the data pattern is not available in the data dictionary, a code for that data is created based on the content of the substring, and the particular code for that phrase is stored in the data dictionary. The code for that phrase is then written to the compressed output stream. When a reoccurrence of a substring is identified in the checkpoint data, the phrase of the substring already stored in the data dictionary is written to the output. Because the phrase value of the code has a physical size that is smaller than the substrings it represents, data compression is achieved. LZW goes beyond most dictionary-based compressors in that it is not necessary to preserve the dictionary to decode the LZW data stream. This can save some amount of space when storing the LZW-encoded data.

The LZW compression algorithm is as follows.

```
w := NIL;
while (there is input)
{
K := next symbol from input;
if (wK exists in the dictionary)
{
w := wK;
}
else
{
```

```
output (index(w));
add wK to the dictionary;
w := K;
}
}
```

Alg.1 LZW compression algorithm

Decompression technique is the reverse process of the compression technique. Decompression reads the encoded data and adds the code to the data dictionary .The encoded data is replaced by string values.

C. System Design

To increase the system capability to tolerate high fault rates, the failed processes should be distributed among the groups. In a modern multiprocessor, multicore nodes, the failure of a single node results in the failure of several processes, so if all the processes of a node belong to the same group, a node failure will lead to several failures within that group. If we distribute all the processes of one node among different groups, then one node failure will lead to one single process failure in several groups, instead of several process failures within the same group.

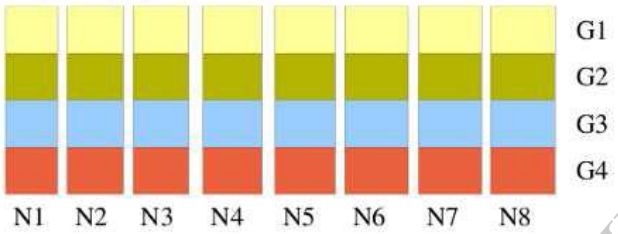

Fig 1. Grouping Strategy

As per Fig.1, we create 4 groups (4 colors) of 8 processors each. When a node fails, each group will have one processor failure, which is the best possible distribution of the failures among the groups. This system can tolerate up to 4 node failures (16 processor failures) and it will be able to recover the lost checkpoints in all the groups.

The encoding process will be significantly faster for 4 groups of 8 processors than for 1 group of 32 processors because the groups can encode in parallel. All the processes in a group must be from different nodes. We will divide the whole system in sectors. A sector is a group of M nodes. A sector is not a group of processes. A sector can be viewed in hardware level as a group of nodes and in a virtual level as a group of groups.

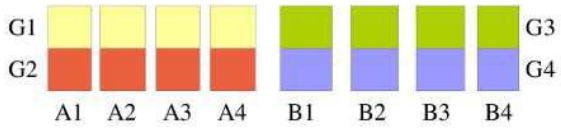

Fig 2.Partitioning the system into sectors and groups.

As per Fig.2 When the groups are built using the presented strategy, the failures will be distributed among the groups then we can use small groups to encode. The failures can affect several nodes, if the groups are smaller than the number of failed nodes, the system will be unable to recover the data. It is important to find a group size that guaranties a good tradeoff between encoding speed and capability to tolerate high fault rates. However, even if the number of failures increases linearly with the system size, the number of nodes affected per failure should not increase that fast. If the average number of nodes affected per failure remains constant, then the group's size can remain constant and then the encoding time and the checkpointing performance will remain constant. Using this model, every supercomputer administrator can choose the size of the groups to tolerate 50% percentage of failures and perform a fast encoding.

IV. CONCLUSION

Diskless checkpointing is an approach that provides high-performance and reliable storage for intermediate or temporary data, such as checkpoint files. To tolerate the less memory space of main memory, a compression method is used. To compress the checkpoint data LZW compression method is used which is a lossless compression algorithm. LZW is dictionary based algorithm which replaces the repeated patterns with dictionary entry. There is no need to pass the dictionary for decompression. This reduces the amount of data to be stored in main memory. By reducing the checkpoint data the network bandwidth also effectively used. To tolerate multiple processor failure the systems are partitioned into groups and failures are distributed in to the groups. During failure the checkpoint data are recovered from the peer processes and the failed system is recovered using that data.

REFERENCES

[1] Z. Chen, G.E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, "Fault Tolerant High Performance Computing by a Coding Approach," Proc. ACM Symp. Principles and Practice of Parallel Programming (PPoPP '05), pp. 213-223, June 2005.

[2] T.-C. Chiueh and P. Deng, "Evaluation of Checkpoint Mechanisms for Massively Parallel Machines," Proc. IEEE Symp. Fault Tolerant Computing (FTCS '96), pp. 370-379, June 1996.

[3] Ge-Ming Chiu, Member, Jane-FerngChiu,(2011) 'A New Diskless Checkpointing Approach for Multiple Processor Failures', IEEE transactions on dependable and secure computing, vol.8, no.4.

[4] J.S. Plank, K. Li, and M.A. Puening, Diskless Checkpointing," IEEE Trans. Parallel Distributed Systems, vol. 9, no. 10, pp. 972-986, Oct. 1998.

[5] L.M. Silva and J.G. Silva, "An Experimental Study about Diskless Checkpointing," Proc. Euromicro Conf. EUROMICRO '98), pp. 395- 402, Aug. 1998.

[6] Doug Hakkarinen and Zizhong Chen," *N*-Level Diskless Checkpointing," IEEE International Conference on High Performance Computing and Communications, 2009.

[7] E. Elnozahy, L. Alvisi, Y. Wang, and D. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACMComputing Surveys, vol. 34, no. 3, pp. 375-408, Sept. 2002.

[8] Leonardo, Naoya, "Distributed Diskless Checkpoints for Large Scale Systems", IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.

[9] Charng-dalu, "Scalable diskless checkpointing for large parallel systems", Phd Thesis, University of Illinois at Urbana-Champaign, 2005.

[10] Z. Chen and J. Dongarra, "A Scalable Checkpoint Encoding Algorithm for Diskless Checkpointing," Proc. IEEE Symp.HighAssurance Systems Eng. Symp. (HASE '08), pp. 71-79, Dec. 2008.