www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 10 October, 2014 Page No. 8589-8591

A new diskless checkpointing approach for handling multiple processor failures

***Dipali B. Parase*[1]*, Dr. Mrs. S. S. Apte*[2]**

[1]Solapur University, Walchand Institute of technology, Solapur, Maharashtra, India
deepali..parase@rediffmail.com

[2]Solapur University, Walchand Institute of technology, Solapur, Maharashtra, India
headcse@gmail.com

Abstract: *In a distributed computing environment, it is necessary to handle the multiple processor failures. In this paper we present a new diskless checkpointing approach which combines neighbor based diskless checkpointing and parity-based diskless checkpointing. As we are storing checkpoint in the peer processors memory, the problem of stable storage is overcome by using neighbor-based diskless checkpointing method. Also for reducing memory consumption problem we use parity-based diskless checkpointing technique. There is no need of dedicated checkpoint processors. It can handle multiple processor failures simultaneously in the system.*

Keywords: Diskless checkpointing**,** failure recovery

1. Introduction

In a system, we run a large computation application. If the system fails, in the middle of its execution, there is need to restart the application after recovery from the failure. When we restart the computation, it will start from the beginning. This results into wasting number of CPU cycles more in computing the same task again. Therefore, most of the systems employ rollback recovery mechanism for failure recovery. In this method we are storing status of a computation onto a stable storage at each regular interval. So, after failure recovery our system should restart from the point where it fails.

Checkpointing using stable storage incurs considerable amount of operational overhead to the system, as number of checkpoint to be taken was restricted. To overcome this we use a diskless checkpointing approach. It contains three methods: neighbor-based, parity based, and Reed-Solomon coding based diskless checkpointing approach.

We use Neighbor-based diskless checkpointing approach and parity based approach.

- **Neighbor-based diskless checkpointing**

In this technique [2], [6], [7], each processor saves its checkpoints in the memory of peer processors. Each checkpoint is stored in its entirety in peer memory, and no coding is involved. Whenever a processor fails, the last checkpoint can be readily recovered from one of these peer processors. However, this approach may consume a large amount of memory to tolerate multiple failures

- **Parity-Based diskless checkpointing**

Parity-based schemes [3], [4] use a dedicated checkpoint processor to store the parity of the checkpoints taken by all the application processors using XOR operations. This approach is simple and easy to implement. Based on some parity array coding technique, two checkpoint processors can tolerate two failures.

Combination of these two techniques is our approach [1]. To overcome the problem of storage overhead we use neighbor based approach. And for reducing memory consumption problem we apply parity based technique.

2. Methodology

Consider, a distributed system consisting of a collection of n processors (or nodes), P0; P1; P2; …; Pn-1, that are interconnected by a (wired or wireless) network. Each processor has physical memory and communication capability. Stable storage installation is not required in the system, and checkpoint data must be stored in the physical memory.

Assume that a computing task is partitioned into n subtasks such that each subtask is executed on a distinct processor Pi, $0 \le i \le n-1$, in a distributed and asynchronous manner. These subtasks communicate with each other by passing messages via the underlying network

The neighbor based diskless checkpointing. Here, we are using two terms CS (Checkpoint Storage node) and CC (Checkpoint Coverage node). Each processor Pi, $0 \le i \le n-1$, must send its checkpoint to a set of at least k other processors for storage. Where, k is size of storage nodes. These processors are called the checkpoint storage nodes (CS) of Pi. Meanwhile, Pi receives checkpoint data from other processors and stores these checkpoints in its (volatile) memory. The processors for which Pi is a checkpoint storage node are called the checkpoint coverage nodes (CC) of Pi. The set of checkpoint storage nodes of Pi are denoted by CSi and the set of checkpoint coverage node of Pi are denoted by CCi, in [1][8].

Initially, we assumed that each processor have their CS and CC list. Each processor in the system stores checkpoint into nodes in CS and receives from its CC's. When processor, say Pi, receives checkpoint from its CCi's it does XORing of own checkpoint and received checkpoint. It also stores copy of its own checkpoint into its own memory.

If processor Pi, fails then it can send the recovery request to its CSi nodes. At least one node in the CSi list must be alive for helping processor Pi in its checkpoint recovery operation. Suppose, say processor Pr is the node in CSi list which is alive. Own checkpoint is stored into checkpoint variable. We also store the previous checkpoint of each processor in the prev_checkpoint variable. And current checkpoint is stored into the curr_checkpoint variable. Then last checkpoint will be calculated by first nullify the previous checkpoint using .checkpoint^=prev_checkpoint and recovered as checkpoint^=curr_checkpoint. Then our system will restart from where it last failed. To handle failure recovery we required only one CS node must be alive.

In [1], the condition for failure recovery was: at least one CS should alive and all CC's of that CS must be alive. But in our case it recovers last checkpoint from CS node. Thus, only one CS node required to recover failure.

.

2. Implementation

We use a distributed or parallel system. Here, we take an application called MAT [8]. This application performs a matrix multiplication. We are having two matrices of size 4000*4000. Consider, we are allowing simultaneous k processor failures in our system. For example k=2. So we require number of processors equal to 5. We take one more processor as master and 5 processors as slaves. Master is for distributing the task and a slave computes the task. Master also informs operator about failed slave. To work in parallel we divide the task of multiplication among processors. After dividing each processor performs its computations. A row is taken as a checkpoint. While performing computation each processor stores its checkpoint into own memory as local copy. Consider the slave's CS and CC lists as in Table 1.

Table 1: List of CS and CC for each slave

Processor (Slave id)	CS	CC
1	{2,3}	{4,5}
2	{3,4}	{5,1}
3	{4,5}	{1,2}
4	{5,1}	{2,3}
5	{1,2}	{4,5}

From the above Table 1 slave 1 stores its checkpoint into CS nodes CS {2, 3}. Consider, slave 1 and slave 3 fails. As k=2, k is maximum number of allowable processor failure. A failed processor is recovered only when it has at least one CS node is alive. Then, we can recover the slave 1 just contacting to the slave 2.and slave 3 is recovered by contacting either slave 4 or slave 5. Since, CS for slave 3 is CS {4, 5}. Hence, our approach handles multiple processor failures [8]

3. Results and Discussion

In our project we are enhancing neighbor-based diskless checkpointing approach. Here, we are considering the combination of neighbor-based diskless checkpointing approach for reducing memory overhead and also for reducing stable storage requirement, we use the parity technique. We are comparing our new approach with the disk-based approach. In the disk-based approach we are storing a checkpoint into a file. Here, we were compared time of storing checkpoint in this disk-based approach with our proposed approach. As shown in Figure 1.

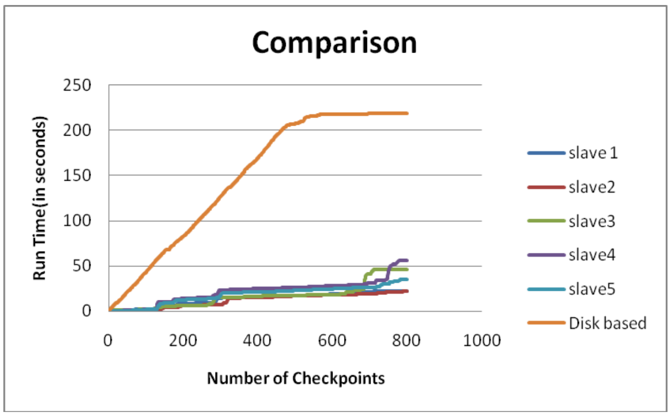

Figure.1 comparison of disk based approach with our proposed approach

From the above comparison we can say that time required for taking checkpoint in our approach is less than in disk-based. Also, we can compare our graph with graph given in paper [1]. From that we clearly understand that the time taken by our scheme and both scheme RS-code and neighbor based diskless checkpointing approach in paper [1], is less.

We have proposed that our approach works for k simultaneous failures. It can handle k simultaneous failures. But the condition is at least one of its node from CS must be alive. Then only we can recover its last status. We compare our approach with approach proposed in [1]. In the paper [1], they compared average time overhead for taking checkpointing in between Reed-Solomon and our approach. The results are shown in the Table 2; in this table we have compared these results with our results.

Table 2: Average Time Overheads (in Seconds) for Checkpoints

Schemes \ #	5	10	15	20	25
RS-Code	3.14	2.64	2.48	2.45	2.43
Results in [1]	1.37	1.38	1.32	1.23	1.22

Disk-based	1.35	3.66	5,79	8.16	9.75
Ours					
Slave 1	0.084	0.38	0.5	0.55	0.64
Slave 2	0.044	0.32	0.40	0.42	0.43
Slave 3	0.18	0.65	0.77	0.85	0.94
Slave 4	0.14	0.28	0.32	0.34	0.37
Slave 5	0.08	0.11	0.15	0.17	0.19

From above results its shows that our approach for taking checkpoint is better than RS-Code, approach in [1], and Disk-based approach.

This study also measures the time it takes to recover from two failures, which is the largest number of simultaneous failures allowed by these schemes. According to our proposed scheme simultaneous processor failure is allowed two i.e. k=2.Table 4 shows the average recovery time for our scheme. In [1] results are shown for average recovery time, to recover previous checkpoints for the three failed processors.

Table 3 Average Recovery Time in Seconds for the Schemes

Schemes	Recovery Time
RS-Code	2.13
Results in paper [1]	0.61
Ours	1.07

4. Conclusion

This study addresses diskless checkpointing issues in a distributed or parallel computing environment and presents a new approach to enhancing neighbor-based schemes to tolerate multiple failures. This method allows checkpoint related operations to be evenly distributed among all processors, achieving good load balance. We have proposed that our approach works for k simultaneous failures. It can handle k simultaneous failures. But the condition is at least one of its node from CS must be alive. Then only we can recover its last status. We compare our approach with approach proposed in [1]. In the paper [1], they compared average time overhead taken in taking checkpointing in between Reed-Solomon and our approach. The results are shown in the Table 2, in this table we have compared these results with our results. From these results it is observed that our approach for taking checkpoint is better than RS-Code, neighbor-based approach in [1], and Disk-based approach.

References

[1] Ge-Ming Chiu, Member, IEEE Computer Society, and Jane-Ferng Chiu, “A New Diskless Checkpointing Approach for Multiple Processor Failures”,IEEE transactions on dependable and secure computing, vol. 8, no. 4, July/august 2011

[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] J.-F.Chiu and G.-M. Chiu, “Hardware-Supported Asynchronous Checkpointing Scheme,” IEE Proc.—Computers and Digital Techniques, vol. 145, no. 2, pp. 109-115, Mar.1998.

[4] J.S. Plank, Y. Kim, and J. Dongarra, “Algorithm-Based Diskless Checkpointing for Fault Tolerant Matrix Operations,” Proc. IEEESymp. Fault-Tolerant Computing (FTCS’95), pp.351-360, June1995.

[5] J.S. Plank, Y. Kim, and J. Dongarra, “Fault-Tolerant Matrix Operations for Networks of Workstations Using Diskless Checkpointing,”J. Parallel Distributed Computing, vol. 43, no. 2, pp. 125-138, 1997

[6] J.S. Plank and K. Li, “Faster Checkpointing with N + 1 Parity,”Proc. IEEE Symp. Fault-Tolerant Computing (FTCS ’94), pp. 288-297, June 1994

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

[8] Parase Dipali B, Dr Mrs. Apte S S, Shegadar21 A R “Enhancement in neighbor based diskless checkpointing approach”, IJECS Volume 3 Issue 9 September, 2014 Page No.7966-7967, September, 2014