International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 9, November- 2012

Handling Multiple Processor Failure Using Diskless Checkpointing Approach

Ms. Dipali B Parase

Walchand Institute of Technology, Solapur, Solapur University, Solapur, Maharashtra, India.

Dr. Mrs. Sulabha Apte

Professor, CSE Department, Walchand Institute of Technology, Solapur, Solapur University, Solapur, Maharashtra, India.

Abstract

In a distributed or parallel computing system a partial failure may easily halt the entire operation of the system. Therefore, many systems employ the checkpoint/ rollback recovery operation. In this paper we are using diskless checkpointing approach. This technique helps to reduce the disk overhead and the memory requirement. To achieve this we are storing the checkpoint of a processor into its set of allowable peer processor and which is in turn responsible for storing the checkpoint for others. The proposed scheme allows failure recovery in a distributed manner.

1. Introduction

Fault tolerance is one of the most desirable properties for many distributed or parallel computing systems. As the number of processors grows, the system's mean time between failures (MTBF) significantly decreases. Normally, processors checkpoints must be stored in disk-based stable storage. Although stable 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. Researchers have devised various techniques to minimize this source of overhead. These techniques include incremental checkpointing, checkpoint buffering with copy-on-write, compression, and memory exclusion. However, the performance of stable storage medium remains a major concern with all of these techniques for the systems.

One method of diskless checkpointing is neighbor-based diskless checkpointing [3]. In this technique, 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. But our aim is to reduce memory requirement. Therefore, we are proposing a new approach to neighbor-based diskless checkpointing that tolerates multiple failures using simple checkpoint and failure recovery operations without relying on dedicated checkpoint processors [2]. In the proposed scheme [1], each processor saves its checkpoints at a set of peer processors and in return is responsible for storing a collection of checkpoints for other peer processors using simple XOR operations. The processor which stores its checkpoint into its peer processor, we call them as storage nodes. And the same processor may acts as storage node for other processor; we call it as coverage node [1].

2. Existing Techniques

Due to storing checkpoint in the stable storage, it increases operational access overhead to the system. Also, the number of checkpoints a system can take is typically restricted. Several techniques have been proposed to reduce the overhead involved in checkpointing and failure recovery operations. Diskless checkpointing approach was first proposed in [6] to avoid the excessive overhead associated with stable storage operation. The various diskless checkpointing techniques that followed can be broadly classified into three categories: neighbor-based, parity-based, and Reed-Solomon code-based.

In the neighbor-based approach, each processor saves its checkpoints in the memory of another processor. When a processor fails, the checkpoint data can be recovered from the corresponding checkpoint processor. There are three different neighbor-based checkpointing schemes: mirroring [3], pairing [3], and ring structured [3]. In the

mirroring scheme, each application processor is assigned a dedicated checkpoint processor in which it stores checkpoints. The pairing scheme organizes application processors into pairs. Each processor sends checkpoints to its partner in the pair and, in return, receives and stores checkpoints for the partner processor. Therefore, dedicated checkpoint processors are not necessary. The ring-structured scheme organizes all processors into a virtual ring. Each processor sends its checkpoints to the following neighbor 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. The parity-based diskless checkpointing technique [5] requires that all application processors coordinate to take checkpoints, with parity data of the checkpoints being saved in the main memory of a dedicated parity processor. When an application processor fails, the dedicated parity processor and all the other processors that are still alive cooperate to decode the last checkpoint for the failed processor. Hence, the amount of diskless checkpoint data to be stored is small in the parity-based technique. However, the time overhead for checkpointing and failure recovery operations depends on the number of application processors in the system. To deal with the scalability problem, some schemes employ a binary tree-based XORing operation for computing checkpoint parity [4]. The Reed-Solomon coding-based approach encodes checkpoints of multiple processors using Galois Field arithmetic. When failures occur in the system, a consistent checkpoint can be restored for each failed processor through the decoding process. However, coding-based techniques involve relatively complex computations for checkpointing and failure recovery.

3. Proposed Scheme

Existing parity-based and Reed-Solomon coding-based techniques generally require extra dedicated checkpoint processors for storing the encoded checkpoint data. thus, we are enhancing neighbor based checkpointing approach using parity and XOR technique. In the neighbor based approach we require storing the checkpoint into the memory of its peer processor and are in turn responsible for storing the checkpoint of other processors. But it requires large amount of memory. To achieve our goal of minimum memory requirement instead of storing the checkpoint into peer processor, we first calculate the parity and this parity is stored into the peer processors memory. If suppose one of the processor fails due to any reason it just sends request to one of its peer processor. Then it will XOR the parity and calculates the checkpoint and sends back to requesting processor [1]. The failure recovery, it can be done only when at least one of its storage nodes should be alive and at least one node should be common in storage node and coverage node. The proposed system architecture is as shown in figure 1.1. Here, it consists of three main parts-1) Process Execution Module 2) Network Module 3) Checkpoint Management Module.

1) Process Execution Module – To take checkpoint at regular intervals we require large computation to be performed. So, in this paper we are dividing a large computation among several processors in the network. Each processor communicates with each other by message passing.

2) Network Module- A network module performs two major tasks. First, it does the node discovery. Second, it sends and receives parity of checkpoint across the network.

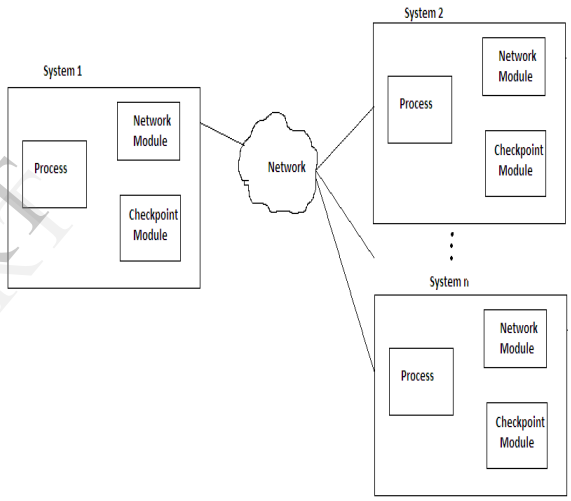


Figure 1.1 System Architecture

3) Checkpoint Management Module- Checkpoint Module is responsible for three major tasks. First, it takes checkpoint as process state and calculates parity. First it stores into its own memory then it sends the calculated parity to its set of peer processor through network module. Second major task of checkpoint module is to if suppose the processor fails, then when processor restarts then this module sends request to its peer processors for getting the parity through network module. Third, major task is to calculate checkpoint from received parity and recover the last status of the process.

4. Data Flow Diagrams

Data flow diagrams for overall execution of a proposed scheme is shown in the figure3.1

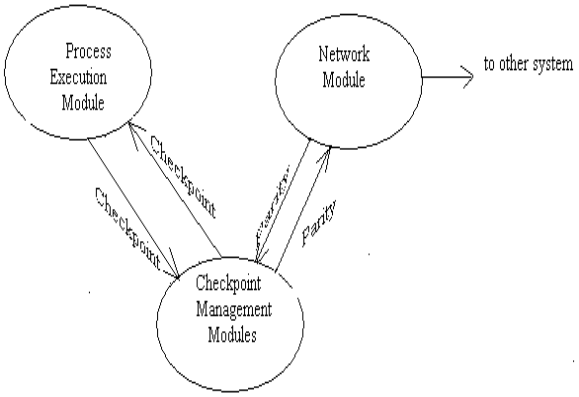


Figure 3.1 Data Flow Diagram for System

Data flow diagram for Process Execution Module is shown in the figure3.2.

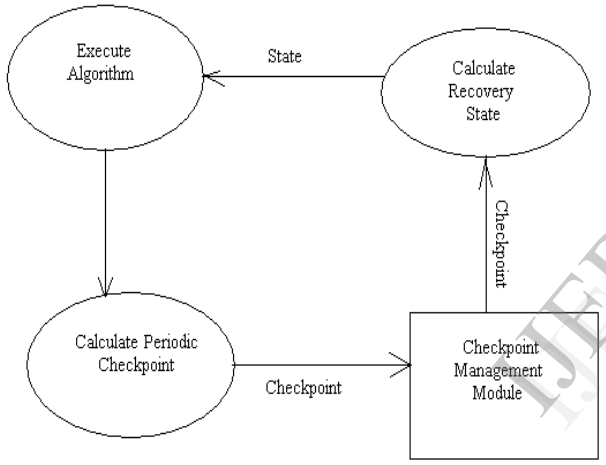


Figure 3.2 DFD for Process Execution Module

Data flow diagram for Checkpoint management Module is shown in the figure3.2

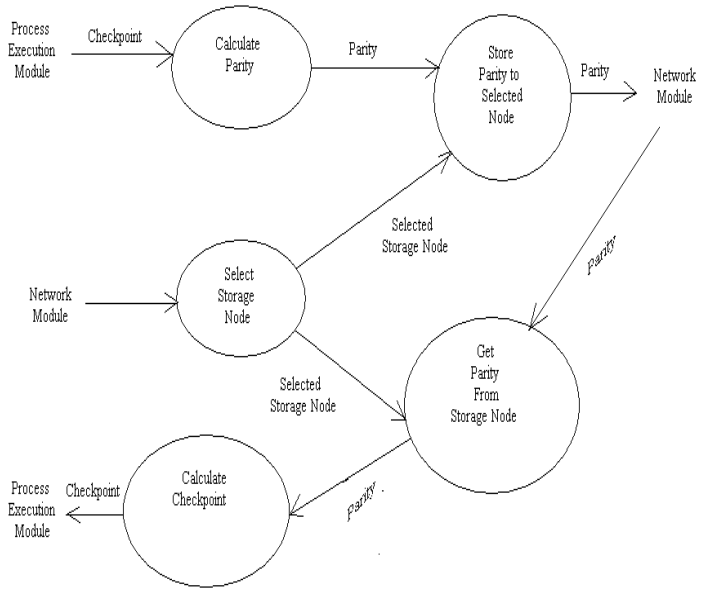
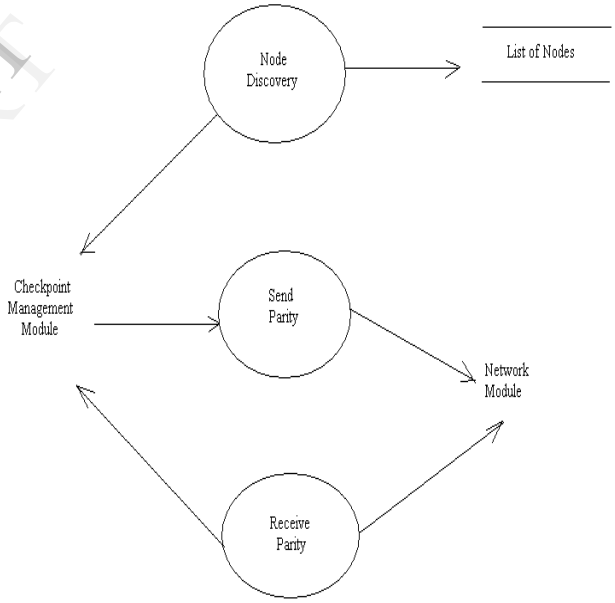


Figure 3.3 DFD for Checkpoint Management Module

Data Flow Diagram for Network Module is as shown in figure 3.4



Fgure 3.4 DFD for Network Module

5. Conclusion

Diskless checkpointing is based on main memory. Also, it is based on coordinated checkpointing, a collection of processors with memories coordinate to have checkpoint of the process state. There are two main memory checkpointing schemes that can be used without hardware changes: neighbor-based checkpointing and parity-based checkpointing. While the neighbor-based checkpointing scheme saves the

checkpoints in the main memory of other processors, parity based checkpointing is based on parity approach.

Thus, we are trying presenting a new approach to enhancing neighbor-based schemes to tolerate multiple failures. It does not require dedicated checkpoint processors. This method allows checkpoint related operations to be evenly distributed among all processors, achieving good load balance.

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

[3] 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.

[4] Z. Chen and J. Dongarra, "Highly Scalable Self-Healing Algorithms for High Performance Scientific Computing," IEEE Trans.Computers, vol. 58, no. 11, pp. 1512-1524, Nov. 2009.

[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.