Checkpoint/Restart System for Private Cloud Development

Win Win Naing, Thinn Thu Naing
University Of Computer Studies, Yangon, Myanmar
winwinnaing89@gmail.com,ucsy21@most.gov.mm

Abstract

Cloud computing provides access to large pool of data, applications and computational resources. Many researchers have been proposed several open-source private cloud management frameworks (e.g., Eucalyptus, Nimbus, and OpenNebula). However, there is no fully automatic fault-tolerance support in private cloud development. In this paper, we propose a new fault-tolerant checkpoint/restart system for hierarchical private cloud. Checkpoint/restart is the simplest way to implement fault-tolerance system in large High Performance Computing (HPC) system. Checkpoint save an application state and restart resume an application execution using the last saved state, on the same machine, or on another machine. We also use Reed-Solomon erasure code to achieve high availability and durability of the checkpoint/restart system.

Keywords - Cloud computing, Fault-tolerance, Erasure Code, Availability, Durability.

1. Introduction

Cloud computing is a subscription-based service which provide computation, software, data access, and storage resources. Cloud users not need to know the location and other details of the computing infrastructure. Since High Performance Computing (HPC) applications requires a lot of resources and cloud computing platform can offers on-demand services with elasticity, scalability on a simple pay-per-use manner, it is widely used in a scientific HPC applications such as medical research, climate modeling, bioinformatics, high-energy and nuclear physics, etc. Another development is the use of cloud computing for HPC is an application is partitioned into a group of tasks for execution on virtualized computing resources. But the number of components on virtualized software or hardware resources can fail at any given moment. Many researchers have proposed many fault tolerant strategies such as user and programmer detection and management, fully automatic and transparent system. However, there is no fully automatic fault-tolerance system in private cloud development such as OpenNebula, Nimbus, Eucalyptus, and OpenStack. All these frameworks do not support tolerated system component failures. Moreover, as no replication support exists, a failure of the frontend node makes the VM management impossible. Therefore, they need to scale with increasing number of resources and continue their operation even though system component failures. For fully automatic and transparent systems, checkpointing and roll back recovery is a promising approach because of its simplicity and lower cost. Checkpoint/restart saves application state at regular intervals and restarts the application form the most recent successful checkpoint after a fault occurs. In order to provide a fault-tolerance system for private cloud development, we propose a checkpoint/restart system for hierarchical designed cloud especially for Eucalyptus.

The overview of this paper is follows. We use coordinated checkpointing protocol to implement fault-tolerance infrastructure with the Reed-Solomon erasure code. The rest of this paper is organized as follows. In the next section, we discuss the related papers with our work. In section 3, we present the checkpointing Theory. And then we describe the Reed-Solomon erasure coding in section 4. In section 5, we present the proposed system. Finally, we conclude the proposed system and present the future work in section 6.

2. Related Work

In this section, we review some paper that is related to the proposed system. Firstly, Eugen Feller, Louis Rilling, and Christine Morin [3] presented the design, implementation and evaluation of a novel scalable and fault-tolerant virtual machine (VM) management framework called Snooze. They utilized a self-organizing hierarchical architecture. They also mentioned that there is no complete fault-tolerance system over private clouds such as OpenNebula [2, 7], Nimbus [7, 8], Eucalyptus [1, 2, 7, 8], and OpenStack [1]. In [6], Long Wang, Karthik Pattabiraman, Lawrence Votta, Christopher Vick, and Alan Wood introduced a coordinated checkpointing model for large-scale supercomputers. Moreover, Sanjay Bansal and Sanjeev Sharma presented many critical factors for checkpointing-based multiple fault tolerance for distributed system in [10]. They discussed many issues of checkpointing based recovery for multiple faults. Their issues are relevant with coordinated checkpointing.

In this paper, we aim to introduce the concept of coordinated checkpointing that is applied in the development of fault-tolerance system for private cloud development. The proposed system combined with Reed-Solomon erasure code to get high availability and durability.

3. Checkpointing Theory

Checkpointing and rollback recovery is the main technique for fault-tolerant system in HPC system. There are two types of technique for rollback recovery: 1) Checkpointing and 2) Log-based protocol.

3.1 Types of Checkpointing

Coordinated checkpoint and uncoordinated checkpoint are the two main techniques used for saving the distributed application execution state and recovering from the system failures.

3.1.1 Coordinated Checkpoint

Coordinated checkpoint is used to save a system-wide consistent state. Coordinate checkpoints are consistent set of checkpoints. These consistent check points are used to bound rollback propagation. Coordinated checkpoint involves the rollback check point of all processes from the last snapshot when a faulty situation is detected, even when a single process crashes. For this reason recovery time is very large and it makes unsuitable for real time applications. In case of frequent failures and multiple faults coordinate check point technique cannot be used. Performance can be improved by decreasing the recovery time. Main reason for large recovery time is restarting all the initial state. Recovery time can be reduced by enabling the restart from last correct state instead of from very first state. There must be some mechanism to ensure restarting from last correct state will reach a state matching the rest of the system, as before the crash. Checkpointing is only taken when all process agree for a consistent state [10].

3.1.2 Uncoordinated Checkpoint with Message Logging

In Uncoordinated checkpoint protocols, all processes execute a checkpoint independently of the others so that recovery can be done independently with each other. Message logging is combined with uncoordinated checkpoint to restart the system from last correct state. It is combined with message logging to ensure the complete description of a process execution state in case of its failure. Besides logging of all received messages, resending the same relevant messages in the same order to the crashed processes during their re-execution is also main function of message logging. By using message logging, uncoordinated checkpointing have fast recovery since restart is neither from the last consistent state nor from the initial state as in case of coordinate checkpointing. Since checkpointing is done independently hence multiple faults can be handled by this approach which cannot be handled by coordinated checkpointing [10].

4. Erasure Code

Some fault-tolerance system uses the replication to achieve high availability and durability for their large system. But replication has some drawbacks. System architects need to increase the number of replicas to achieve high durability for their system. Then, the increase in the number of replicas increases the bandwidth and storage requirements of the system. Therefore, we decided to use erasure code to increase system availability. An erasure code [4, 9] provides redundancy without the overhead of strict replication. Erasure code divide an object into m fragments and recode them into n fragments, where n>m. We call r=m/m<1 the rate of encoding. A rate r code increases the storage cost by a factor of 1/r. The key property of erasure code is that the original object can be reconstructed form any m fragments. Figure 1 shows one example for erasure code. In this figure, k represents the source data and n represents the encoded data.

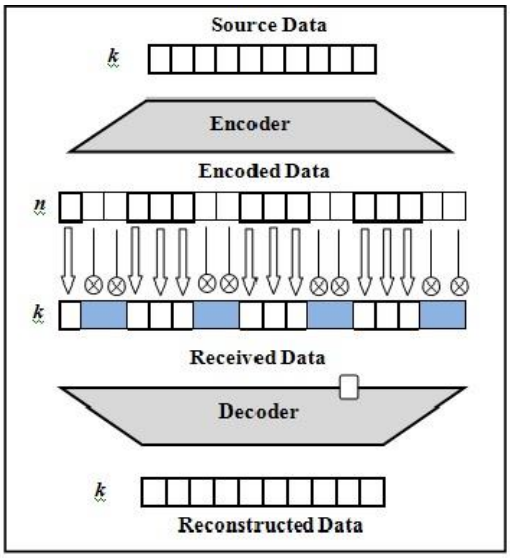


Figure 1. The Erasure Coding Example

Moreover, there are two most popular types of erasure codes. They are (1) Reed-Solomon coding, and (2) Low-Density Parity-Check (LDPC) coding. But, we only explore Reed-Solomon coding in this paper.

4.1 Reed-Solomon Coding

In checkpoint/restart system, we must use a stable storage to store checkpoint file. The stable storage must be fault tolerance. Therefore, we develop checkpoint/restart system by combining with Reed-Solomon [8, 5] method. The basic operation is as follows. Reed-Solomon Coding use the *Vandermonde matrix* to calculate and maintain checksum words.

$$c_i = F_i(d_1, d_2, \ldots, d_n) = \sum_{j=1}^{n} d_j f_{i,j} \quad (1)$$

where F_i is a linear combination of the data words.

$$FD = C. \quad (2)$$

where D and C are the data and checksum words vectores.

When we define F to be the m*n Vandermonde matrix: $f_{i,j} = j^{i-1}$, and thus the above equation becomes:

$$C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \vdots & \vdots & & \vdots \\ f_{m,1} & f_{m,2} & \cdots & f_{m,n} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} = FD \quad (3)$$

A distribution matrix is employed so that the check blocks are calculated from the vector with m dot products.

To explain recovery from errors, Reed-Solomon uses the Gaussian Elimination: AD=E

$$AD = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ 1 & 2 & & n \\ \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & \cdots & n^{m-1} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = E \quad (4)$$

where $A = \begin{pmatrix} I \\ F \end{pmatrix}$ and $E = \begin{pmatrix} D \\ C \end{pmatrix}$.

Moreover, Galois Field arithmetic is used so all elements have multiplicative inverses. Then the act of decoding is straightforward.

Given any n of the data and check blocks, a decoding matrix may be derived from the distribution matrix, and the remaining data blocks may be calculated again with dot products.

5. The Proposed System

In this section, we present a new checkpoint/restart system for private cloud development. We use coordinated checkpoint protocol to implement the proposed system. Because coordinated checkpoint protocol is more consistent and less overhead than uncoordinated checkpoint protocol. We also use Reed-Solomon coding instead of using replication because replication has the potential to increase availability and durability. Therefore, there are two main parts in the proposed system. They are (a) Coordinated Checkpoint Phase and (b) Erasure Code Phase. Figure 2 shows the proposed system architecture.

5.1 Coordinated Checkpoint Phase

The proposed system based on the hierarchical cloud infrastructure such as Eucalyptus. In Eucalyptus, cloud controller (CLC) performs as a main entry point of the system and cluster controller (CC) implements about high level scheduling decision over node controllers (NCs). Therefore, we aim to implement coordinator node on cloud controller (CLC). By doing this, we can

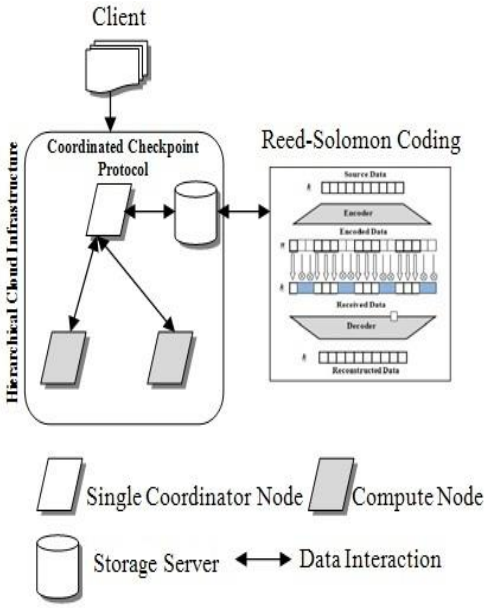
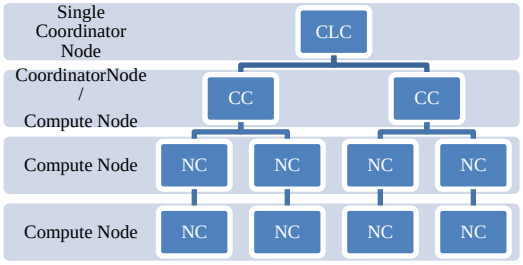


Figure 2. The Proposed System Architecture

manage and synchronize checkpointing information of all cluster controller (CC) and node controllers (NCs).

Coordinated checkpoint [6] includes a single coordinator node, or master node and many slave nodes. Therefore, we assume cloud controller (CLC) as a single coordinator node and cluster controllers (CC) and node controller (NC) as the compute nodes. Figure 3 shows the coordinated checkpointing architecture for hierarchical cloud infrastructure based on Eucalyptus. It includes the following high-level components according to the figure 3.



CLC---Cloud Controller
CC---Cluster Controller
NC---Node Controller

Figure 3. Coordinated Checkpoint Architecture for Hierarchical Designed Private Cloud Development

- **Node Controller (NC)** controls the execution, inspection, and terminating of VM instances on the host where it runs.
- **Cluster Controller (CC)** gathers information about and schedules VM execution on specific node

controllers, as well as manages virtual instance network.

- **Cloud Controller (CLC)** is the entry-point into the cloud for users and administrators. It queries node managers for information about resources, makes high-level scheduling decisions, and implements them by making requests to cluster controllers.

5.1.1 Coordinated Checkpoint Protocol

In coordinated checkpoint phase, a single coordinator node and multiple compute nodes are included. The single coordinator has a responsibility to get a consistent set of checkpoints for multiple compute nodes. The checkpoint file contains the information such as the stack, heap, registers of the process, the status of pending signals, signal handlers, accounting records, and terminal state, etc. Moreover, we assume our system based on Eucalyptus architecture. Therefore, the single coordinator (CLC) must maintain the information of multiple compute nodes (CCs) and node controllers (NCs) such as application states, data states, resource states etc. Then, the application or system state and a set of checkpoint are stored to the storage server. The following steps of coordinated checkpointing protocol are based on the basic principles of coordinated checkpointing [6].

Step (1): The single coordinator node broadcast an initial 'requests' message to all the compute nodes.

Step (2): When all compute nodes receive 'request' message from the coordinator node, they stop all its activities at a consistent state and reply the ready message to the coordinator node.

Step (3): After receiving 'ready' from all the compute nodes, the coordinator node broadcasts 'checkpoint' message to all the compute nodes.

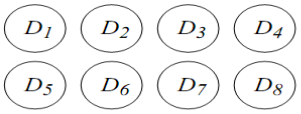
Step (4): When each compute node receives 'checkpoint' message, they leave their state and send a 'done' message to the coordinator node.

Step (5): When the coordinator collects the 'done' messages from all the compute nodes, it broadcasts 'proceed' messages to all the compute nodes.

Step (6): On receiving 'proceed' message each compute node begin to write the checkpoint to the file system and continues its activity from the point at which it requested.

5.2 Erasure Codes Phase

After storing the application state and checkpointed information to the storage server, we perform the Reed-Solomon erasure code phase on that information. We gain high availability and durability by performing the Reed-Solomon coding on it. The following example shows the configuration using this technique for n=8 data and m=2 checksum devices.



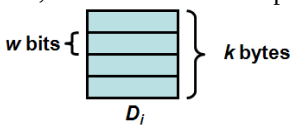
$$C_1 = F_1(D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8)$$

$$C_2 = F_2(D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8)$$

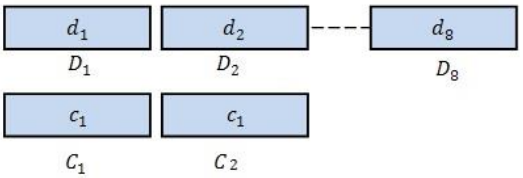
$$D_j = Data\ Device$$

$$C_i = Checksum\ Device$$

The coding on the checksum node C_1 and C_2 is computed by using functions F_1 and F_2 from Eq. (1) respectively. Then, each device breaks up into word:



And then, size of each word is w bits. In this example, we assume each device holds just 1 word:



When m =1 node fails, we can calculate C_1 by taking the parity (XOR) of the data words from Eq. (1), (2) and (3):

$$c_1 = F_1(d_1, \ldots, d_8) = d_1 \oplus d_2 \oplus \ldots \oplus d_8$$

If data word d_3 of D_3 fails, we can reconstruct by restoring the parity of the corresponding words on the remaining devices from Eq. (4):

$$d_3 = d_1 \oplus d_2 \oplus d_4 \oplus \ldots \oplus d_8 \oplus c_1$$

Once the data words are reconstructed, we can reconstruct the checksum words from the data words and F. Therefore, the whole system is reconstructed.

6. Conclusions and Future Work

In this paper, we presented a new fault-tolerant system by using coordinated checkpointing and Reed-Solomon erasure code. The major conclusions from this paper include: a new fault-tolerant system for private cloud development especially for hierarchical architecture. Another fact is use of the Reed-Solomon erasure code instead of using replication to achieve high availability and durability. In the future, we plan to perform a performance comparison of our checkpoint/restart system with existing fault-tolerant system for existing open-source cloud management system.

References

[1] D. Nurmi, R. Wolski, C. Grzegorczyk, “Eucalyptus: A Technical Report on an Elastci Utility Computing Architecture Linking Your Programs to Useful Systems”, UCSB Computer Science Technical Report, 2008.
[2] D. Nurmi, R. Wolski, C. Grzegorczysk, “The Eucalyptus Open-source Cloud-computing System”, University of Califonia, Santa Barbars, 2009.
[3] E. Feller, L. Rilling, C. Morin, “Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds”, December 2011.
[4] H. Weatherspoon, John D. Kubiatowics, “Erasure Coding vs. Replication: A Quantitative Comparison”, University of California, Berkeley.
[5] J. S. Plank, “A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems”, University of Tennessee, July 19, 1996.
[6] L. Wang, K. Pattabiraman, L. Votta, C. Vick, A. Wood, “Modeling Coordinated Checkpointing for Large-Scale Supercomputers”, University of Illinois at Urbana-Champaign, Sun Microsystems.
[7] P. Sempolinski, D. Thain, “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus”, University of Notre Dame.
[8] P. Takako Endo, G. Estacio Goncalves, J. Kelner, D. Sadok, “A Survey on Open-source Cloud Computing Solutions”, Universidad Federal de Pernambuco.
[9] R. L. Collins, James S. Plank, “Assessing the Performance of Erasure Codes in the Wide-Area”, University of Tennessee.
[10]S. Bansal, S. Sharma, “Identification of Critical Factors in Checkpointing Based Multiple Fault Tolerance for Distributed System”, Medi-Caps Institute of Technology and Management, Indore, India.