International Journal of Advance Research in Science and Engineering
Volume No.07, Issue No.04, April 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

Study of Checkpoint Restore mechanism for Fault Tolerance in Cloud computing

Pooja Kathalkar[1], A. V. Deorankar[2]

[1] *Department of Computer Science and Engineering,*
Government College of Engineering Amravati (MH), India
[2] *Department of Computer Science and Engineering,*
Government College of Engineering Amravati (MH), India

ABSTRACT

Fault tolerance is probably one of the greatest challenges that the high performance computing community is facing today. As a part of this study, we understand the many faces of checkpoint / restore, the de-facto fault tolerance method in practice. Checkpointing is an efficient fault tolerant component that takes a snap intermittently to save redundant process execution in the memory in the event of task or VM failure. This is particularly valuable in spot instances circumstances as it save unfinished computation should in case there is process failure and the client do not have to pay for that. Checkpointing policies are taken occasionally at a client defined rate. On the one hand, checkpointing overhead instances are taken into consideration. On the other hand, the price of saving checkpoints is not taken into account, as the cost of saving service is insignificant in contrast to cost of VMs. In addition, it can be performed simultaneously with the job execution; therefore the period of time for transmitting checkpoint data is unnoticed as it is negligible.

Keywords : *Checkpoint Restore, Cloud Computing, Fault Tolerance, multi-level rollback, VM*

1. Introduction

Fault tolerance in cloud computing is a critical requisite for realtime systems, due to potentially devastating consequences of faults when it occurs in both heterogynous and homogeneous computing environments. The fault tolerance mechanism presented in this research article consists of the following mechanism: fault detector, checkpointing and restart mechanism, job migration and replication mechanism. To implement this design, the following assumptions have to be made.

- The datacenter is anticipated to made enough cloud resources, to circumvent VM denials as a result of resource disputation. The assumption is not over ambitious as the resources needed are much lesser than the datacenter capability.
- At any given period, at most one host or VM can fail, which will result to job migration. It can also be finished or executed by its backups before another host fails.
- Faults can either be temporary or stable and also self-regulating from one another.

1.1 Process Checkpointing

The goal of process checkpointing is to save the current state of a process. In current HPC applications, a process consists of many user-level or system-level threads, making it a parallel application by itself. Process checkpointing techniques generally use a coarse-grain locking mechanism to interrupt momentarily the execution of all the threads of the process, giving them a global view of its current state, and reducing the problem of saving the process state to a sequential problem. Independently of the tool used to create the checkpoint, we distinguish three parameters to characterize a process checkpoint:

At what level of the software stack it is created;

• How it is generated;

• How it is stored.

1.1.1 Saving Executive State

The checkpoint routine, provided by the checkpointing framework, is usually a blocking call that terminates once the serial file representing the process checkpoint is complete. It is often beneficial, however, to be able to save the checkpoint in memory, or to allow the application to continue its progress in parallel with the I/O intensive part of the checkpoint routine. To do so, generic techniques, like process duplication at checkpoint time can be used, if enough memory is available on the node: the checkpoint can be made asynchronous by duplicating the entire process, and letting the parent process continue its execution, while the child process checkpoints and exits. This technique relies on the copy-on-write pages duplication capability of modern operating systems to ensure that if the parent process modifies a page,the child will get its own private copy, keeping the state of the process at the time of entering the checkpoint routine.

1.1.2 Restoring Executive State

When a failure has occurred, the recovery mechanism restores system state to the last checkpointed value. This is the fundamental idea in the tolerance of a fault within a system employing checkpoint-recovery. Ideally, the state will be restored to a condition before the fault occurred within the system. After the state has been restored, the system can continue normal execution.

State is restored directly from the last complete snapshot, or reconstructed from the last snapshot and the incremental checkpoints. The concept is similar to that of a journaled file system, or even RCS(revision control

system), in that only the changes to a file are recorded. Thus when the file is to be loaded or restored, the original document is loaded, and then the specified changes are made to it. In a similar fashion, when the state is restored to a system which has undergone one or more incremental checkpoints, the last full checkpoint is loaded, and then modified according to the state changes indicated by the incremental checkpoint data.

If the root cause of the failure did not manifest until after a checkpoint, and that cause is part of the state or input data, the restored system is likely to fail again. In such a case the error in the system may be latent through several checkpoint cycles. When the it finally activates and causes a system failure, the recovery mechanism will restore the state (including the error!) and execution will begin again, most likely triggering the same activation and failure. Thus it is in the system designers best interest to ensure that any checkpoint-recovery based system is fail fast - meaning errors are either tolerated, or case the system to fail immediately, with little or no incubation period.

Such recurring failures might be addressed through multi-level rollbacks and/or algorithmic diversity. Such a system would detect multiple failures as described above, and recover state from checkpoint data previous to the last recovery point. Additionally, when the system detects such multiple failures it might switch to a different algorithm to perform its functionality, which may not be susceptible to the same failure modes. The system might degrade its performance by using a more robust, but less efficient algorithm in an attempt to provide base level functionality to get past the fault before switching back to the more efficient routines.

1.2 Classification Of Checkpointing Schemes

Checkpointing can be classified along various dimensions. We will start by identifying checkpoints by their scope. These are the typical granularities at which checkpointing schemes are developed, each with their own advantages and disadvantages.

1.2.1System-Level Checkpoints

These checkpoints are taken at the OS level. The thought behind these checkpoints is to provide fault tolerance to applications that are already running, but do not have any application level fault handling mechanism. We believe these are best suited for machines that either run a bunch of very varied application workloads and its very difficult for all of them to maintain checkpoints independently or have legacy applications running on them that are not equipped with checkpointing. The advantage of these checkpoints are that applications (and thus their developers) need not care about checkpointing recovery, which can be a daunting task to implement.

1.2.2 Application-level Checkpoints

These checkpoints are at the other extreme. They are completely application-based, and are in fact, written by the developers of the application. The notion behind this scheme is that the application

itself is the best judge of when and what it needs to save in order to minimize loss in the face of failure. Advantage(s) - In agreement with the notion, the application has precise knowledge of the state it needs to save in order to reconstruct following a failure. There might be large amounts of memory that need not be saved at all (which the system-level checkpointing schemes are not at all aware of),andthatcouldresultinveryefficientcheckpointing.Disadvantage(s)-Neitherare all the already running, widespread legacy applications equipped with checkpointing code, nor is it possible to update them to include checkpointing facilities.

1.2.3 Library-level Checkpoints

This checkpointingmechanismisalsoreferredtoascompiler-levelorruntime-level.In contrast to the two extreme granularities mentioned above, library-level conveniently sits at a spot neither too close to the application, nor too far away from it. In terms of benefits, these checkpointing schemes identify pretty closely with the application that is running and can do a fair bit of analysis to identify important information of an application at compile-time / runtime. While this seems to give the best of both worlds,one of the biggest challenges that this technique faces, is when to take acertain checkpoint. A typical case where timing of the checkpoint can make a huge difference, is when checkpoints are initiated when a process is in a loop, or is performing some temporarycomputationthatisverymemoryorI/Ointensive.Iftheschemehadbeena little more application aware, it would have prevented this by taking a checkpoint at a more appropriate time. Some example of library-level checkpointing are [James Plank et al. 1995], [Michael Litzkow, Todd Tannenbaum, Jim Basney and Miron Livny 1997] and [Yi-min Wang et al. 1995]. Another way of classifying checkpoints is on the amount of data stored ateachcheckpoint interval.

1.2.4 Non-Incremental (Whole) Checkpoints

In non-incremental checkpoints, the entire memory (the bulk in any checkpoint) is saved to disk at every interval. This is beneficial when most of the memory is dirtied ineveryinterval.Mostworkloadsshowthatthisisnottruebecauseoflocality.Another advantage of this technique is that it is only essential that you store the latest checkpointondisk.Insystemswherediskspaceisverylimitedand/orcostly,thistechnique proves more cost effective than incremental checkpointing. The obvious disadvantage of this scheme is that entire memory needs to be written to disk on every interval, an inherently costly task. For intervals that are very far apart, this scheme might make sense, but for sub-second intervals, it is a near impossible scheme to implement.

1.2.5 Incremental Checkpoints

As the name suggests, this technique only saves the changed pages of memory (since the last checkpoint) onto disk. Locality aids this technique tremendously. Unlike entire checkpoints, we need to store the whole sequence of checkpoints taken from the start till the latest increment in order to reconstruct the system in case of failure. Due to this, not only is more disk space needed, but reconstruction cost is also higher than non-incremental checkpoints.

But, since recovery is not the common case, this technique still fares better than non-incremental checkpoints, especially in read-intensive workloads. Another disadvantage of this technique is that the memory is updated at word-granularity and memory is checkpointed at page-granularity.

2. Related Work

A survey of the literature on fault tolerant checkpointing shows that a large number of papers have been published.

The Chandy-Lamport [1] algorithm is one of the earliest nonblocking all-process coordinated checkpointing algorithm for static nodes. In this algorithm, markers are sent along all channels in the network which leads to a message complexity of O(N2), and requires channels to be FIFO.

Lai and Yang [2] proposed an algorithm. In this algorithm, when a process takes a checkpoint, it piggybacks a flag to the message it sends out from each channel. The receiver checks the piggybacked flag to see if there is a need to take a checkpoint before processing the message. If so, it takes a checkpoint before processing the message to avoid an inconsistency. To record the channel information, each process needs to maintain the entire message history on each channel as part of the local checkpoint.

Elnozahy et al. [3] proposed an all-process nonblocking synchronous checkpointing algorithm with a message complexity of O(N). They use checkpoint sequence numbers to identify orphan messages, thus avoiding the need for processes to be blocked during checkpointing.

Koo-Toeg [4] proposed a minimum-process coordinated checkpointing protocol which relaxes the assumption that all communications are atomic. It reduces the number of synchronization messages and number of checkpoints.

Cao and Singhal [5] proposed minimum-process blocking algorithm for mobile systems. Every process maintains its direct dependencies in a bit array of length n for n processes. Initiator process collects the direct dependency vectors of all processes, computes minimum set. After that, it broadcasts the checkpoint request along with the minimum set to all processes.

Kim and Park [6] proposed an improved scheme to address failures during checkpointing. It allows the new checkpoints in some subtrees to be committed. In the approach, a process commits its tentative checkpoint if none of the processes, on which it transitively depends, fails; and the consistent recovery line is advanced for those processes that committed their checkpoints. The initiator and other processes which transitively depend on the failed process have to abort their tentative checkpoints. Thus, in case of a node failure during checkpointing, total abort of the checkpointing is avoided.

Neves et al. [7] gave a loosely synchronized coordinated checkpointing protocol that removes the overhead of synchronization. This approach assumes that the clocks at the processes are loosely synchronized. Loosely synchronized clocks can trigger the local checkpoints at all the processes roughly at the same time without a coordinator. After taking a checkpoint, a process waits for a period, which is sum of maximum time to detect a failure of other process in the system and the maximum deviation between clocks. It is assumed that all checkpoints

belonging to a particular coordination session have been taken without the need of exchanging any message. If a failure occurs, it is detected within the specified time and the protocol is aborted.

L. Kumar et al. [8] proposed an all-process non-intrusive checkpointing protocol for distributed systems, where just one bit is piggybacked along with normal messages. This is done by incurring extra overhead of vector transfers during checkpointing.

L. Kumar et. al [9] and P. Kumar et. al [64] reduced the height of the checkpointing tree and the number of useless checkpoints by keeping nonintrusiveness intact, at the extra cost of maintaining and collecting dependency vectors, computing the minimum set and broadcasting.

Higaki and Takizawa [10] proposed a hybrid checkpointing protocol, where fixed hosts checkpoint synchronously and MHs checkpoint independently. Mobile stations use message logging and checkpointing, while fixed stations use only checkpointing, to form a consistent global state.

In 2017, Shafi'i Muhammad Abdulhamid, Muhammad Shafie Abd Latiff proposed Checkpointed league championship algorithm[11] in which they develop a fault-tolerance aware task scheduling scheme for the IaaS Cloud technology using a Checkpointed League Championship Algorithm (CPLCA) intelligent scheme. The task migration is combined with the checkpointing strategy in this scheme. The motivation being that while job migration can help to transfer failed jobs to any available VM, the checkpointing strategy will help to continue the execution from the last saved state. This will ultimately help to reduce the time taken to restart the execution from the beginning instead.

3. Conclusion

Checkpoint - Rollback is a technique which can be used to build fault tolerance into a computing system.

This paper provide information about checkpoint Restore technique and various mechanism proposed by different author to enhance the performance of checkpoint restore technique. This study throws light on process of check pointing and classification of check pointing schemes.

References

[1]. Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," ACM Transaction on Computing Systems, vol. 3, No. 1, pp. 6375, February 1985.

[2]. T.H. Lai and T.H. Yang," On Distributed Snapshots", Information Processing Letters, vol. 25, pp. 153-158, 1987.

[3]. Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.

[4]. Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987

[5]. Cao G. and Singhal M., "On the Impossibility of Minprocess Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.

[6]. J.L. Kim, T. Park, "An efficient Protocol for checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, pp. 955-960, Aug. 1993.

[7]. Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.

[8]. L. Kumar, M. Misra, R.C. Joshi, "Checkpointing in Distributed Computing Systems" Book Chapter "Concurrency in Dependable Computing", pp. 273-92, 2002.

[9]. L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003

[10]. Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.

[11]. Shafi'i Muhammad Abdulhamid, Muhammad Shafie Abd Latiff., "A checkpointed league championship algorithm-based cloud scheduling scheme with secure fault tolerance responsiveness", Applied Soft Computing Journalhttp://dx.doi.org/10.1016/j.asoc.2017.08.048

[12]. Saurabh Kadekodi, "Compression in Checkpointing and Fault Tolerance Systems", . ACM Trans. Embedd. Comput. Syst. V, N, Article A (January YYYY), 8 pages.

[13]. Rachit Garg, Praveen Kumar, "A Review of Fault Tolerant Checkpointing Protocols for Mobile Computing Systems", International Journal of Computer Applications (0975 – 8887) Volume 3 – No.2, June 2010.