Local Disaster Recovery Using Virtualization Technology

Aye Myat Myat Paing
University of Computer Studies, Yangon
paing.ayemyat@gmail.com

Abstract

Business applications running on IT infrastructure necessitate high levels of availability in order to minimize the amount of downtime experienced during any planned and unplanned outages. As a result, disaster recovery has gained great significance in IT. Exploiting virtualization and ability to automatically reinstall a host, where the action on a virtual machine is performed only when a disaster occurs. Virtualization affords significant cost and performance advantages over more traditional disaster recovery options such as tape backup or imaging. Our approach is to design and implement a continual migration strategy for virtual machines to achieve automatic failure recovery. By continually and transparently propagating virtual machine's state to a backup host via live migration techniques, trivial applications encapsulated in the virtual machine can be recovered from hardware failures with minimal downtime while no modifications are required. Moreover, our framework intends to monitor virtual machines for problems such as CPU utilization, I/O activity, and memory utilization. This raises a difficult problem, since it is quite difficult to discriminate based on these measures between a virtual object that is performing properly, and one that is quite ill. We apply the out-of-band monitoring using virtualization and machine learning can accurately identify faults in the guest OS, while avoiding the many pitfalls associated with in-band monitoring.

Keywords: virtualization, availability, fault-tolerance, machine learning

1. Introduction

Virtualization has been widely adopted by data centers for transparent load balancing, application mobility, server consolidation and secures computing [1]. With one physical machine hosting many virtual machines, a single node failure may result in more severe disruption to hosted services, which brings great challenge for automatic failure recovery in virtualized computing environments. One of the most general solutions for failure recovery is to replicate the states of the protected virtual machine to a backup host, with which the virtual machine may be recovered from host failures. Although virtual machine replication can be done in various ways, its consistency and efficiency are not guaranteed. Most virtual machines maintain memory and external storage states in separate ways, which may result in an inconsistency when replicated to the backup host. In the mean time, fast and transparent failure recovery requires the backup to be synchronized with the primary virtual machine; however, synchronizing on every change brings too much performance tradeoffs [9].

In this paper, we present a continual migration mechanism for virtual machine based fault tolerant systems by continually and transparently propagating virtual machine's states to a backup host via live migration [3] techniques. While other migration based high available systems [9] maintains a secondary disk image file on the backup host, continual migration propagates disk states via network attached storage, which is more common in a data center configuration and provides better agility in both configuration and deployment. Some research work achieves failure recovery by taking and restart from checkpoints [7], [4], [8]. Continual migration aims to provide transparent fault tolerance for commodity applications in data centers or virtual computing environments [5].In addition, our framework make use of out-of-band monitoring system using machine learning and virtualization. Traditional approaches use in-band monitoring agents. However in-band agents suffer from several drawbacks: they need to be written or customized for every workload (operating system and possibly also application), they comprise potential security liabilities, and are themselves affected by adverse conditions in the monitored systems. Therefore, this paper describes one approach to out-of-band monitoring that performs this discrimination based on statistical analysis, as implemented using machine learning.

2. Related Work

Remus provides an extremely high degree of fault tolerance, to the point that a running system can transparently continue execution on an alternate physical host in the face of failure with only seconds of downtime, while completely preserving host state such as active network connections. This system encapsulates protected software in a virtual machine, asynchronously propagates changed state to a backup host at frequencies as high as forty times a second, and uses speculative execution to concurrently run

the active VM slightly ahead of the replicated system state [2].

Ta-Shma [8] uses a continuous data protection (CDP) enabled file system to preserve virtual machine disk image files. In a CDP enabled file system, every write operation is logged and revertible, which enables the virtual machine to move to any historical states for debugging, intrusion detection or fault tolerance. Though historical data is valuable for system analysis, it is not quite suitable for fault tolerant purpose due to tremendous storage overheads.

Kemari [10] achieves virtual machine replication by event driven synchronizations. Unlike Remus, Kemari migrates disk images via network attached storage, which may result in an inconsistency when failure happens. Kemari solves this problem by synchronizing virtual machine's memory state every time before disk operations are issued. However, when running I/O intensive workloads, Kemari suffers a severe performance penalty due to high frequency synchronizations.

Machine learning has been applied to problem determination for complex systems in avionics and energy production and generation. For computer systems machine learning has been applied primarily in the domains of security and performance management. Service management products that aid virtualization (used to monitor virtual machines out-of-band) with machine learning (used to analyze the monitoring results). The importance of virtualization as a framework for software rejuvenation was explored by Silva et al. [6].

3. Continual Migration Mechanism

The basic concept of continual migration is to continually and transparently migrates virtual machine's states to the backup host until failure is detected. For most virtual machines, two parts of states are required to migrate: the internal state such as memory, registers and/or internal device buffers, and external states such as attached hard disk images. In a live migration, internal states are propagated via migration data stream transferred through network, while external states are shared in network attached storage. With such configuration, virtual machines are able to migrate between hosts within local network rapidly and agilely, without needing to pre-copy large disk image files. Continual migration adopts this configuration but extends the migration protocol so that states can be transferred continually to backup hosts.

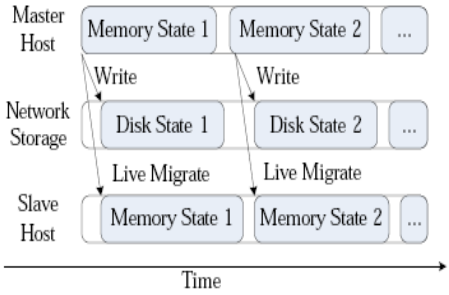

Fig. 1. Continual migration mechanism

The major difference between live and continual migration is that the virtual machine on the target host is not guaranteed to be consistent during a live migration, while continual migration must ensure that even an ongoing migration is interrupted by a failure on the source, there must be a valid and consistent state on the backup system to recover from. This brings three major challenges to our work:

3.1. Migration Continuity

Although live migration provides minor machine downtime, its overall migration time is significant. To achieve fast and frequent checkpoints, continual migration must reduce overall migration time to a matter of tens of milliseconds. Besides, migration procedures should be changed so that ongoing migrations will not corrupt previous migrated consistent internal states on the backup host.

3.2. External State Consistency

Continual migration participants share disk image via network attached storage, thus the consistency of the image file must be guaranteed even if cached disk data fails to flush to disk as failure happens.

3.3. Coherent Consistency

When applications are rolled back to the last migrated state, the corresponding disk image may have been changed, which will result an incompatibility between memory and disk states on the backup host. Continual migration must ensure that whenever a failure happens, the backup host can load a consistent virtual machine with compatible internal and external states.

4. Internal State Migration

In continual migration, internal states are replicated via live migration data stream. Continual migration extends live migration with following modifications:

Scheduled migration. The major change in continual migration is to periodically issue outgoing migrations. This is done by scheduling a migration immediately after previous migration is completed. One of the most simple and straightforward strategies is to issue migrations with static time intervals.

Lightweight migration. Compared to live migration protocols, continual migration introduces a lightweight migration strategy for reducing migration iteration time. For the first live migration iteration, whole memory blocks are transferred the same way as traditional migration protocols, while after the migration is completed, continual migration continues to track dirtied pages, as following live migration iterations will only transfer dirtied pages.

Buffered migration. Continual migration buffers migration data to preserve consistent internal states. On the source host, migration data is buffered until the current migration iteration is completed, and then this piece of migration stream is transferred to the backup host together with the length and checksum. Incoming migration streams on the backup host are buffered and verified before merging into migrated states, so that incomplete migration data will not affect the current consistent states.

5. Three-phase Commits

When failure happens, the internal states are rolled back to last migrated states; however, the external states in the network attached storages needs more discussing. By buffering write operations in memory, buffered block device ensures the consistency inside the disk image itself; however, the time to flush memory buffer to external image files is crucial. When failure happens, internal states are rolled back to the last migration state, while external states in the network attached storage won't, which may result in an incompatibility between memory and disk states on the recovered virtual machine. Continual migration introduces a Three-phase commits mechanism for this kind of coherent consistency problem, as shown in Figure 2.

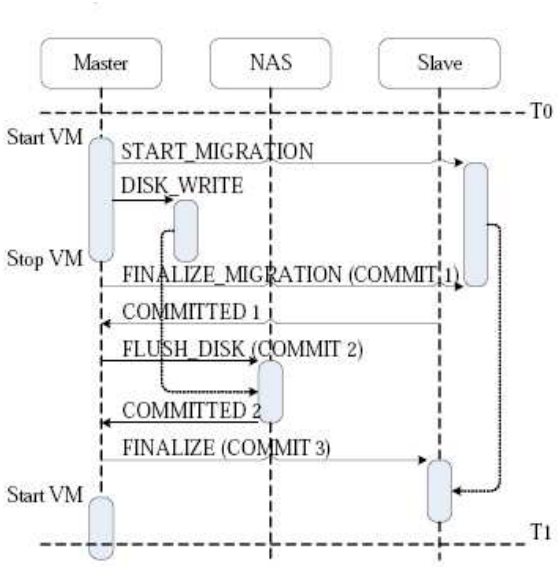

Fig. 2. Three-phase commits

We consider a time designated as T0, when both the source and the destination holds a consistent virtual machine states. In the first phase, live migration is issued to the backup host, when memory and processor updates are copied to the backup host and buffered for verification. The attached buffered block device is switched to async mode, in which the write operations are buffered without affecting the states in the network attached storage. When this round of live migration comes to an end, a COMMIT 1 message is posted through the event channel, to which the backup host will response to complete the first phase. After receiving the first COMMIT response, the source switches the disk to sync mode to flush buffer back to disks. This is considered as the second phase. In the last phase, a COMMIT 3 message is posted to the backup host, upon which the backup host will merge buffered internal states to the backup virtual machine. Both the source and the destination now enter a new consistent state designated as T1.

Three-phase commits ensures that at any time, the backup host can find a consistent virtual machine state to recover from failure. For failures between T0 and commit 2, both the internal states and external states are untouched, thus backup virtual machine can be easily recovered to T0 state. For failures between commit 2 and commit 3, external states are updated to newer version, while complete internal state updates have already been transferred to the backup host, with which the internal states can be updated to match the external states. Failures after commit 3 won't affect backup consistency since both internal and external states are updated to T1.

6. Out-of-band Monitoring System

We will implement a two-stage system using machine learning. First, we generate a *classifier*, which is a piece of code that can identify faults. Second, the classifier is applied to data from live virtual objects to determine their current state. In particular, we first collect data from a variety of virtual machines under various loads. This data is then fed to a machine learning process, which outputs the classifier. The classifier is a decision routine that can label each observation, and do so in a manner that is generally consistent with its training data. In the second stage, the system is running and the observations are collected as before. But this time, each observation is fed through the classifier and a label is predicted. If the label matches an actionable condition (such as an imminent system failure), the system can be configured to take the appropriate action. This process corresponds to a discipline of machine learning known as "supervised learning". The supervision comes in the form of the labels that are attached to the initial observations. In our system, there will be two possible labels: normal and faulty.

7. Conclusion

We have described the design of our hypervisor based fault tolerant systems via continual migration mechanism. It can reduce machine downtime, while maintaining tolerable performance tradeoffs. Additionally, we intend to implement the out-of-band monitoring system for detecting failure in virtual machines using virtualization and machine learning algorithm. This monitoring system can accurately identify faults in the guest OS, will be our future work.

References

[1] A. Agbaria and R. Friedman, "Virtual-machine-based heterogeneous checkpointing," *Softw. Pract. Exper.*, vol. 32, no. 12, pp. 1175–1192, 2002.

[2]Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield
Remus: High Availability via Asynchronous Virtual Machine Replication.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286.

[4] G. Vallee, T. Naughton, H. Ong, and S. L. Scott, "Checkpoint/restart of virtual machines based on xen," in *High Availability and Performance Computing Workshop (HAPCW06)*, 2006.

[5] J. Huai, Q. Li, and C. Hu, "Research and design on hypervisor based virtual computing environment," *Journal of Software*, vol. 18, no. 8, pp. 2016–2026, 2007.

[6] L. M. Silva, J. Alonso, P. Silva, J. Torres, and A. Andrzejak. Using virtualization to improve software rejuvenation. In *IEEE International Symposium on Network Computing and Applications (IEEE-NCA)*, Cambridge, MA, USA, July 2007.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 2003, pp. 164–177.

[8] P. Ta-Shma, G. Laden, M. Ben-Yehuda, and M. Factor,
"Virtual machine time travel using continuous data protection and checkpointing," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 1, pp. 127–134, 2008.

[9] T. C. Bressoud and F. B. Schneider, "Hypervisor-based fault tolerance," in *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM, 1995, pp. 1–11.

[10] Y. Tamura, K. Sato, S. Kihara, and S. Moriai, "Kemari: virtual machine synchronization for fault tolerance," in *USENIX '08 Poster Session*, San Jose, CA, USA, 2008.