Pre-copy Based Live Migration Using Lossless Compression Algorithm

Ei Phyu Zaw
University of Computer Studies, Yangon
zaw.eiphyu@gmail.com

Abstract

Virtualization is a methodology of logically dividing computer resources. By emulating a complete hardware system, from processor to network card, each virtual machine can share a common set of hardware. It allows multiple virtual machines, with heterogeneous operating systems to run side by side on the same physical machine. Migration operation system instance across distinct physical hosts is a useful tool for administrators of data centers. . Live migration is done by performing most of the migration while the operating system is still running, achieving very little downtime. By carrying out the majority of migration while OSes continue to run, we achieve impressive performance with minimal service downtime and total migration time. In this paper, we propose the design, implementation, and evaluation of pre-copy based live migration using lossless compression algorithm for virtual machines (VMs) across a Gigabit LAN.

1. Introduction

Today's IT departments are under increasing pressure to manage and support expanding computer resources while at the same time reducing costs. Virtualization technology, which lets multiple operating systems run concurrently on the same physical server, has become a broadly accepted method to meet these requirements. By converting under-utilized physical servers into virtual machines that run on a single physical server, organizations can reduce space, power and hardware costs in the data center. Because virtual machines are generally much faster to recover in a disaster than physical computers are, virtualization also increases server uptime and reliability.

A virtual machine is a software implementation of a machine (computer) that executes programs like a real machine. It was originally defined as "an efficient, isolated duplicate of a real machine". An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the VM. Virtualization is popular, particularly among the data center and cluster computing communities. Migrating operating system instances across distinct physical hosts is a useful tool for data centers administrators. It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance.

The software layer providing the virtualization is called a Virtual Machine Monitor (VMM) or hypervisor. From Fig.1, the virtualization platform is installed directly onto the computer's hardware provide a platform on which one or more virtual machines using hypervisor architecture. It can be created with a unique guest operating system and its own set of applications installed It virtualizes all of the resources of a physical machine, thereby defining and supporting the execution of multiple virtual machines [13].

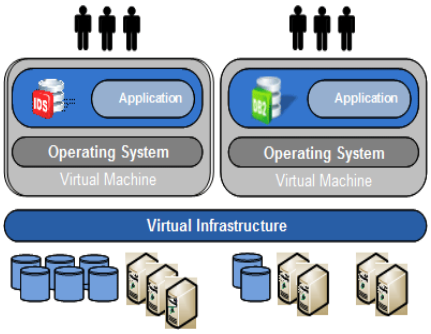

Figure1. Hypervisor Virtualization Architecture

In a VM-based cluster, multiple virtual machines share a physical resource pool. Due to dynamically varying workloads, some nodes are often under-utilized, while others may become heavily-loaded. In various application scenarios, VM migration is expected to be fast and VM service degradation is also expected to be low during migration. Live migration is a key feature of system virtualization technologies. In this paper, we focus on VM migration within a cluster environment where a network-accessible storage system (such as SAN or NAS) is employed. Only memory and CPU status needs to be transferred from the source node to the target one. Live migration techniques in the state of the art mainly use pre-copy approach which first transfers all memory pages and then copies pages just modified during the last round iteratively. VM service downtime is expected to be minimal by iterative

copy operations but total migration time is prolonged. The above issue in pre-copy approach is caused by the significant amount of transferred data during the whole migration process.

This paper presents a novel approach to optimize live virtual machine migration based on pre-copy algorithm. We first use memory compression to provide fast VM migration. Virtual machine migration performance is greatly improved by cutting down the amount of transferred data.

The rest of this paper is organized as follows. Section 2 describes the related work. In section 3, we discuss the live VM migration. Section 4 the proposed system is described. Finally section 5 concludes the paper.

2. Related Work

This section briefly describes some of the different architectures which have implemented virtual machine migration techniques.

Xen supports Live Migration [3]. It is a useful feature and natural extension to virtualization platforms that allow for the transfer of a VM from one physical machine to another, with little downtime of the services hosted by the VM. Live migration transfers the working state and memory of a VM across the network, while they are running.

Xen also supports high performance VM migration by using Remote Direct Memory Access (RDMA) [9]. It offers performance increase in VM migration by avoiding TCP/IP stack processing overhead. RDMA implements a different transfer protocol, where origin and destination VM buffers must be registered before any transfer operations, reducing it to "one sided" interface. Data communication over RDMA does not need to involve CPU, caches, or context switches. This allows migration to be carried out with minimum impact on guest operating systems and hosted applications.

Zap [2] supports transparent migration of legacy and networked applications. Zap provides a thin virtualization layer on top of the operating system that introduces a *Process Domain (pod)* abstraction. A pod represents a process group with the same virtualized view of the system and a private namespace. This virtualized view associates virtual identifiers with OS resources such as PIDs and network addresses. This decouples processes in a pod from host dependencies, and forms the basic unit of migration.

Internet Suspend-Resume (ISR) technique [4] looks to exploit temporal locality, as memory states are likely to have considerable overlap in the suspended and the resumed instances of the VM. Temporal locality refers to the fact that the memory states differ only by the amount of work done since the VM was last suspended before being initiated for migration. To exploit temporal locality each file in the file system is represented as a tree of small sub-files. A copy of this tree exists in both the suspended and resumed VM instances. Predictably, the downtime (the period during which the service is unavailable due to there being no currently executing instance of VM) is high, compared to some of other techniques.

The approach best suited for live migration of virtual machines is *pre-copy*. These include hypervisor-based approaches from VMware[6], Xen[3], KVM[8], OS-level approaches that do not use hypervisors from OpenVZ[9]. Pre-copy technique incorporates iterative push phases and a stop-and-copy phase which lasts for a very short duration. In short, the pages to be transferred during round *'n'* are only the ones dirtied during round *'n-1'*.

Two different pre-copy techniques have been implemented over the Xen: *Managed Migration* and *Self Migration* [3]. In case of managed migration, the migration is performed by the daemons running in the management VMs of the source and the destination. These daemons are responsible for creating a new VM on the destination machine, and coordinating transfer of live system state over the network. In the initial round, all the pages are transferred and subsequently only those pages that were dirtied in the previous rounds (as indicated by a *dirty bitmap*) are migrated. Xen uses *shadow page tables* to log dirty pages [5].

Another novel strategy post-copy is also introduced into live migration of virtual machines [11]. In this approach, all memory pages are transferred only once during the whole migration process and the baseline total migration time is achieved. But the downtime is much higher than that of pre-copy due to the latency of fetching pages from the source node before VM can be resumed on the target.

Techniques for reducing the downtime during memory migration have been suggested over Capsule [10]. These include optimization features which employ demand-paging, along with a clever usage of an algorithm known as *ballooning.* The algorithm eliminates or pages out less useful data in a system, which can be implemented in the virtual memory manager of the OS. Ballooning helps in reducing the size of the compressed memory state and this reduces the start-up time after migration. Certain disadvantages may arise in cases where the pages holding critical cached data, dirty buffers or active data are immediately used after migration.

3. Live VM Migration

Virtual machine migration takes a running virtual machine and moves it from one physical machine to another. This migration must be transparent to the guest operating system, applications running on the operating system, and remote clients of the virtual machine.

Live Migration migrate OS instances including the applications that they are running to alternative virtual machines freeing the original virtual machine for maintenance. It rearranges OS instances across virtual machines in a cluster to relieve load on congested hosts without any interruption in the availability of the virtual machine.

A key challenge in managing the live migration of OS instances is how to manage the resources which include networking, storage devices and memory.

Networking: In order for a migration to be transparent all network connections that were open before a migration must remain open after the migration completes. To address these requirements we observed that in a cluster environment, the network interfaces of the source and destination machines typically exist on a single switched LAN.

Storage devices: We rely on storage area networks (SAN) or NAS to allow us to migrate connections to storage devices. We assume that all physical machines involved in a migration are attached to the same SAN or NAS server. This allows us to migrate a disk by reconnecting to the disk on the destination machine.

Memory: Memory migration is one of the most important aspects of Virtual machine migration. Moving the memory instance of the VM from one physical state to another can be approached in any number of ways. But traditionally, the concepts behind the techniques tend to share common implementation paradigms. Migrating memory, which can be in the range of hundreds of megabytes to a few gigabytes in a typical system today, needs to be done in an efficient manner.

The memory migration in general can be classified into three phases:

Push phase: The source VM continues running while certain pages are pushed across the network to the new destination. To ensure consistency, the pages modified during the transmission process must be re-sent.

Stop-and-copy phase: The source VM is stopped, pages are copied across to the destination VM, and then the new VM is started.

Pull phase: The new VM starts its execution, and if it accesses a page that has not yet been copied, this page is faulted in, across the network from the source VM.

4. The Proposed System

The Logical steps that we execute when migration an OS are summarized in Figure 2. We take a conservative approach to the management of migration with regard to safety and failure handling. We view the migration process as a transactional interaction between the two hosts involved:

- **Step-1**: A request is issued to migrate an OS from host A to host B. We initially confirm that the necessary resources are available on B and reserve a VM container of that size.

- **Stage-2**: During the first iteration, all pages are transferred from A to B using lossless compression algorithm. Subsequent iterations copy only those pages dirtied during the previous transfer phase.

- **Stage-3**: We suspend the running OS instance at A and redirect its network traffic to B. CPU state and any remaining inconsistent memory pages are then transferred. At the end of this stage there is a consistent suspended copy of the VM at both A and B. The copy at A is still considered to be primary and is resumed in case of failure.

- **Stage-4**: Host B indicates to A that it has successfully received a consistent OS image. Host A acknowledges this message as commitment of the migration transaction, host A may now discard the original VM, host B becomes the primary host and now activated.

This approach to failure management ensures that at least one host has a consistent VM image at all times during migration. It depends on the assumption that the original host remains stable until the migration commits, and that VM may be suspended and resumed on that host with no risk of failure.

The main contribution of the paper is that we design and implement a novel approach, memory compression techniques based on pre-copy to minimize the total migration time of live migration.

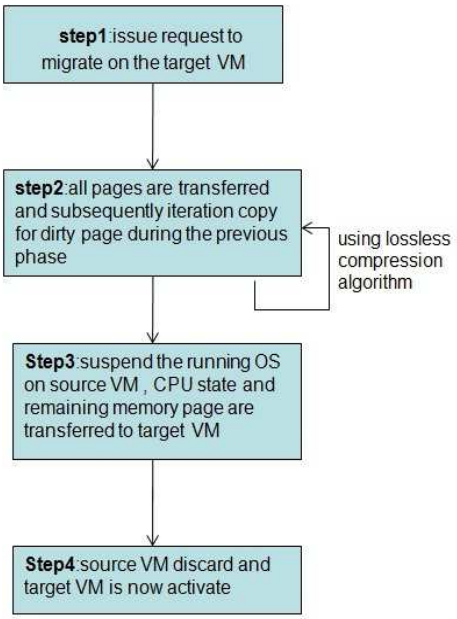

Figure 2. The design overview of our proposed system

Compression technique can be used to significantly improve the performance of live migration. The compression algorithm is first lossless because the compressed data need to be exactly reconstructed. Second, the overhead of memory compression should be as small as possible. If the overhead outweighs the advantage of memory compression, live VM migration would

not get any benefit from it. A compression algorithm with low overhead is ordinarily simple but difficult to achieve high compression ratio. So,

how to balancing the relationship between overheads and compression effects is crucial to design a good algorithm for live VM migration.

We consider the design options for migration OSes running services with the **Lemple-zip Encoding compression algorithm** [1]**.** LZ encoding is known as dictionary-based encoding. The idea is to create a dictionary (table) of strings used during the communication session. The compression algorithm extracts the smallest substring that cannot be found in the dictionary from the remaining non-compressed string. It is based on the notion that data occur repeatedly in the message being compressed. It is also a *lossless* compression algorithm. So, if we compress data using the algorithm, and then decompress the compressed version, the result will be an exact copy of the original data.

The LZ compression algorithm is shown in Figure3.

1. Set the *current-location pointer* to the beginning of the message.
2. Find the longest match that occurs in the *search window* for the beginning sequence of characters in the *lookAhead window*.
3. Process the possible match:
   a. If there is no match for even the first character in the *lookAhead window*, output a tuple containing the following three values and increment the *current-location pointer* by 1:
      offset: 0
      stringLen: 0
      nextChar: The unmatched first character in the *lookAhead window*.
   b. If there is a match, output a tuple containing the following three values and increment the *current- location pointer* by the length of the match plus 1:
      offset: The location of the beginning of the match in the *search window* relative to the current location.
      stringLen: The length of the match.
      nextChar: The first non-matching character in the *lookAhead window* following the characters that match.
4. Go back to step 2 and repeat the process until the end of the message is encountered, saving the collection of output tuples in some appropriate container.

Figure 3. LZ lossless compression algorithm

5. Conclusion

By integrating live OS migration into the virtual machine monitor (VMM), we enable rapid movement of interactive workloads within clusters and data centers. Live migration makes it possible to use VMs with less effort and greater flexibility than before. These benefits translate to time and money savings in any virtual machines.

In this paper, we have proposed the design of pre-copy based live migration which uses data compression technique. Because the smaller amount of data is transferred, the total migration time and downtime will be both decreased significantly. Service degradation will also be decreased greatly. So, the end-to-end time of the migration and the impact on VM running on the machines involved in the migration can be controlled by managing the memory resource.

6. References

[1] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23(3):337–343, 1977.

[2] S. Osman, D. Subhraveti, G. Su, and J. Nieh, *The Design and Implementation of Zap: A System for Migrating Computing Environments*, 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. July, C. Limpach, I. Pratt, and A. Warfield, *Live Migration of Virtual Machines*, Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation, 2005

[4] M. Kozuch, M. Satyanarayanan, *Internet Suspend/Resume*, Fourth IEEE Workshop on Mobile Computing Systems and Applications, Callicoon, NY, June 2002.

[5] 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*, Proceedings of the Nineteenth ACM Symposium Operating System Principles (SOSP19), pages 164.177. ACM Press, 2003.

[6] LIM, B.-H., AND HUTCHINS, G. Fast transparent migration for virtual machines. In *Usenix, Anaheim, CA* (2005), pp. 25–25

[7] M. Nelson, B. Lim, and G. Hutchines, "Fast transparent migration for virtual machines," in Proceedings of the USENIX Annual Technical Conference (USENIX'05), 2005, pp. 391–394.

[8] KIVITY, A., KAMAY, Y., AND LAOR, D. kvm: the linux virtual machine monitor. In *Proc. of Ottawa Linux Symposium* (2007).

[9] OPENVZ. *Container-based Virtualization for Linux, http://www.openvz.com/*.

[10] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, *Optimizing the Migration of Virtual Computers*, Proceedings of the 5th Symposium on Operating Systems Design and Implementation, Boston, Massachusetts, USA, December 9–11, 2002

[11] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in Proceedings of the ACM/Usenix international conference on Virtual execution environments (VEE'09), 2009, pp. 51–60

[12] B. Barham , B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, "Xen and the art of virtualization," In Proc. the 19th Int. ACM Symp. Operating Systems Principles, New York, USA, ACM Press, 2003, pp.164-177.

[13] R. J. Creasy, *The Origin of the VM/370 Time-sharing System*, IBM J. RES. DEVELOP., Vol. 25, No. 5, September 1981.

[14] R. Goldberg, "Survey of virtual machine research," IEEE Computer, pp. 34–45, Jun. 1974.

[15] M. Ekman and PerStenstrom, "A robust main-memory compression scheme," in Proceedings of the International Symposium on Computer Architecture (ISCA'05), 2005, pp. 74–85..