Working Set Prediction for Live Virtual Machine Migration

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

Abstract

Live migration of virtual machines has been a powerful tool to facilitate system maintenance, load balancing, fault tolerance, and power-saving in clusters or data centers. In pre-copy approach that is mainly used in live migration, VM service downtime is expected to be minimal but total migration time is prolonged by iterative copy operations and the significant amount of transferred data during the whole migration process. In this paper, we present a framework that includes pre-processing phase, push phase and stop phase, for reducing the amount of transferred data in pre-copy based live migration. In pre-processing phase of the framework, we propose Least Recently Used (LRU) based working set prediction algorithm in physical memory of virtual machine. The proposed algorithm classify working set list that collects the most recent used memory pages and inactive list that collects the least recent used memory pages. Applying the proposed working set prediction algorithm, we can reduce the amount of iteration time of push phase and reduce the total migration time. From calculation result compared with pre copy based live migration that is not include pre-processing phase, the proposed framework can reduce the total migration time.

Keywords:*Least Recently Used (LRU), Migration Time, Pre-copy based live migration, Virtual Machine*

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. Virtualization is provide by better utilizing computing resources**,** improving scalability, reliability and availability while lowering total cost of ownership.

Migrating operating system instances across distinct physical hosts is a useful tool for virtualization technology. It allows a clean separation between hardware and software, and facilitates fault management.

Live migration of virtual machines is a useful capability of virtualized clusters and data centers. It can be done by performing while the operating system is still running. It allows more flexible management of available physical resources by making to load balance and do infrastructure maintenance without entirely compromising application availability and responsiveness. VM migration is expected to be fast and VM service degradation is also expected to be low during migration.

The key challenge is to achieve impressive performance with minimal service downtime and total migration time in live migration[6]. During the migration, resource in both machines must be reserved on migration application and the source machine may not be freed up for other purpose and so it is important to minimize the total migration time. It need to consider the moving of Virtual Machine's memory content, storage

content, and network connections from the source node to the target node.

The best technique for live migration of virtual machines is pre-copy [3]. It incorporates iterative push phases and a stop-and-copy phase which lasts for a very short duration. By 'iterative', pre-copying occurs in rounds in which the pages to be transferred during round n are those that are modified during round n-1. The number of rounds in pre-copy migration is directly related to the working set which are being updated so frequently pages. The final phase stop the virtual machine, copies the working set and CPU state to the destination host.

The issue of pre-copy based live migration is that total migration time is prolonged [8]. It is caused by the significant amount of transferred data during the whole migration process and maximum number of iterations must be set because dirty pages, frequently updated page, are ensured to converge over multiple rounds.

In this paper, we propose the algorithm to predict the working set, the collection of recent used memory pages, in pre-copy based migration for virtual machines and then define working set list and inactive list. We also present the proposed framework for pre-copy based live migration to reduce the total migration time. The results from the calculation we can reduce total migration of live migration of virtual machines.

The rest of this paper is organized as follows. Section 2 describes the related work. In Section 3, we discuss the live migration of virtual machines. In Section 4, the proposed framework of pre-copy based live migration and calculation result are described. Finally, Section 5 concludes the paper.

2. Related Work

Pre-copy [3] is the prevailing live migration technique to perform live migration of VMs. This technique keep downtime small by minimizing the amount of VM state that needs to be transferred during downtime but it can reduce pre-copy's effectiveness and increase total migration time during migration process because pages that are repeatedly dirtied may have to be transmitted multiple times.

CR/TR-Motion [10] is proposed to transfer checkpoint and execution trace files rather than memory pages in pre-copy phase to provide fast VM migration because the total size of all log files is much less than that of dirty pages. It can reduce total migration time and downtime of migration but CR/TR-Motion is valid only when the log replay rate is larger than the log growth rate.

W. Huang et al.[16] uses remote direct memory access (RDMA) to transfer VM migration traffic. This system offers performance increase in VM migration by avoiding TCP/IP stack processing overhead. It implements a different transfer protocol, where origin and destination VM buffers must be registered before any transfer operations, reducing it to "one sided" interface and does not need to involve CPU, caches, or context switches in data communication.

In post-copy live migration of virtual machines [8], 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.

A typical operating system will have a number of free pages. At the beginning of a migration cycle, the OS can return some or all of these pages to reduce the time taken for the pre-copy migration phase in post-copy live migration [8]. However if the contents of these pages are needed again, they will need to be faulted back in from disk, incurring greater overall cost.

H. Jin et al. [14] introduce memory compression technique (MECOM) into live pre-copy based VM migration. Based on memory page characteristics, MECOM design a particular memory compression algorithm for live migration of VMs. Because the smaller amount of data is transferred and only very low compression overhead is introduced, the total migration time and downtime are both decreased significantly but compression time is

performance bottleneck of additional overhead introduced by compression operations.

3. Background Theory

3.1. Live Migration of Virtual Machines

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 as shown in Figure 1.

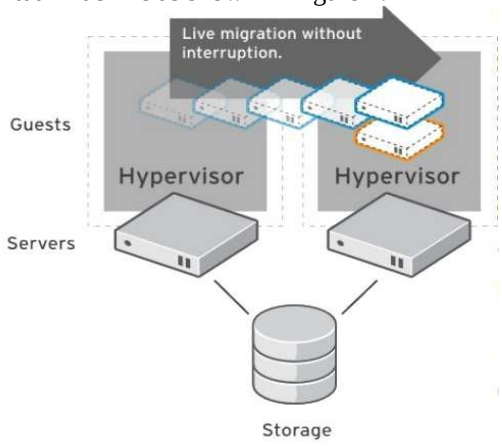

Figure 1. Live Migration of Virtual Machines

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, 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. 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.

The memory migration in general uses one or two phases from the following:

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.

In pre-copy based live migration of virtual machine, it involves a bounded iterative push phase and then a typically very short stop-and-copy phase. It first transfers the memory pages iteratively, in which the pages modified in a certain round will be transferred later in the next round. The iterative push phase continues until stop conditions. After that, the source VM stops and transfers its own state and modified pages from the last iteration.

Many hypervisor-based approaches such as VMware [7], Xen [3] and KVM [1] is used the pre-copy approach for live migration of virtual machines. OpenVZ [9] also use this approach for OS-level migration.

3.2. Memory Management in Hypervisors

When running a virtual machine, the hypervisors creates a contiguous addressable memory space for the virtual machine. This allows the hypervisor to run multiple virtual machines simultaneously while protection the

memory of each virtual machines from being accessed by others. The hypervisor must protect its memory regions from modification by guest OSes. Each time a guest OS requires a new page table, because a new process is being created, it allocates and initializes a page from its own memory reservation and registers it with the hypervisor and all updates must be validated by the hypervisor. The hypervisor distinguishes between two kinds of memory.

Machine memory: It is the whole physical memory in the system.

Pseudo-physical memory: It is an abstraction valid for each domain, allowing the guest OS to consider it's own memory as a contiguous block, hiding the underlying potentially fragmented physical layout (machine memory).

3.3.Least Recently Used (LRU) Replacement Algorithm

It associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. The problem is to determine and order for the fames defined by the time of last use. Two implementations are feasible: Counters and Stack. We apply Stack implementation that keeps a stack of page numbers and most recent memory page move to the top. So, the memory pages used in the recent past is always on the top of the stack. By dynamically monitoring memory accesses and constructing the LRU list, we can predict the Working Set list of a virtual machine.

4.Proposed Framework for Pre-copy based Live Migration

In this Section, we present the proposed framework for pre-copy based live migration to reduce the total migration time. The proposed framework consists of the pre-processing phase, push phase and stop and copy phase as shown in Figure 2.

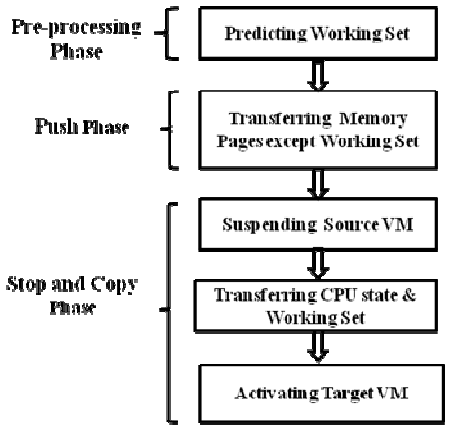

Figure 2. Proposed Framework for pre-copy based live migration

(1) Pre-processing Phase

The system applies the proposed working set prediction algorithm as the pre-processing phase. This algorithm is based on Least-recently-used (LRU) replacement algorithm and collects the most recent used memory page as the working set list and collects the least recent used memory pages as the inactive list.

(2) Push Phase

The system transfer memory pages except working set list in first iteration and then memory pages modified during the previous iteration are transferred to the destination.

(3) Stop and Copy Phase

This phase consists of three steps. We suspend the source virtual machine for a final transfer round. We transfer last modified pages and CPU state as and then discards the source VM and activates the Target VM.

The design involves iteration though multiple rounds of copying in which the VM memory pages that have been modified since the previous copy are resent to the destination.

The main contribution of the paper is that we propose a working set prediction algorithm to minimize the amount of iterations and reduce the total migration time.

In the source VM, we apply the prediction algorithm for working set as the pre-processing step. This algorithm predict and define the working set list which is the collection of memory pages in future use to reduce not only the amount of modified pages during the push phase but also the rounds of copying. Prediction time is performance bottleneck of addition overhead introduced by prediction operation. We use Least Recent Used (LRU)-based prediction algorithm to run prediction operation without a notable execution time.

4.1. Working Set Prediction Algorithm

In pre-copy based live migration, the system 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 that caused by the significant amount of transferred data during the whole migration process.

We propose the working set prediction algorithm in the framework which can reduce the amount of transferred memory pages in pre-copy based migration process. The proposed algorithm predict the most recently used memory pages that directly affects the total migration time

In the proposed algorithm as shown in Figure 3, memory of virtual machine is divided into two lists where the most recently used pages are placed in the Working Set list while least recently used pages are in inactive list.

When a new page is faulted into main memory, it is placed in the Working Set list. If the Working Set list is full, the least recently used page is moved into the inactive list to place the most recent used page. If the inactive list is full that is the memory of virtual machine is full, the least recent page of inactive list is discarded. The Working Set list is managed by least recently used (LRU) algorithm. The inactive list is ordered by each page's time of removal from the Working Set list. When any reference to the inactive list forces to cause a page fault, the VM moves it into the Working Set list.

```
if (Request page j)
{
    if (! Working Set list is full)
        {
          place the page j in the Working Set using the
          linear order;
        }
     else
     {
        if (request page j is not in Working Set)
            {
                remove page i whose is located at the
                end of the Working Set to the inactive
                list;
                insert request page j at the top of the
                Working Set;
            }
        else
            {
                find request page j in Working Set and
                move to the top of Working Set;
            }
    }
}
```

Figure 3. Working Set Prediction Algorithm

We organize the Working Set list as a doubly linked list where each node has two pointers. Whenever a memory page is requested, three operations will be performed on the Working Set list. Firstly, the proposed algorithm finds the requested page in the Working Set list. If the requested page is already exist in Working Set list, move the requested page to the top of the Working Set list. If the requested page is not exist in the Working Set list, delete the page which locate at the end of the Working Set list and insert the requested page at the top of the Working Set list.

4.2. Calculation Results

In this section, we describe the calculation of the total migration time in the propose framework and pre-copy based live migration. Assuming that the memory could be processed in the propose framework as pages with a fixed size equal to $\boldsymbol{P}$ bytes, $\boldsymbol{M}$ is the memory size, the bandwidth available for the transfer is constant and equal to $\boldsymbol{B}$ bytes per second, $\boldsymbol{W_S}$ is the working set list and $\boldsymbol{W_M}$ is the modified page per second as shown in Table1.

Table 1. Abbreviations for the Proposed Framework

P	Page Size
M	Memory Size
B	Transfer Rate (bytes per second)
W_S	Size of Working Set list
W_M	Modified Pages per second
T_i	Execution time of i[th] iteration for *push phase*
T_{stop}	Execution time for *stop phase*
T_{pre}	Execution Time for *preprocessing phase*

Therefore, the time cost to transfer the memory pages in the proposed framework is as follows.

Execution time of i[th] iteration for *push phase* of the proposed framework:

$$T_i = \frac{(M_i - W_S)P}{B} \quad (1)$$

where i=1,2,3,…n and $M_n \leq W_M$ and $M_1 = M$ for first iteration and then modified memory page size is updated by:

$$M_{i+1} = T_i W_M \quad (2)$$

Execution time for *stop phase* of the proposed framework:

$$T_{stop} = \frac{W_S P}{B} \quad (3)$$

Total Migration Time of Proposed Framework $= T_{pre} + \sum_i^n T_i + T_{stop}$ (4)

And the time cost to transfer the memory pages in pre-copy based live migration is as follows.

Execution time of i[th] iteration for *push phase* of pre-copy based live migration:

$$T_i = \frac{M_i P}{B} \quad (5)$$

where i=1,2,3,…n and $M_n \leq W_M$ and $M_1 = M$ for first iteration and then modified memory page size is updated by:

$$M_{i+1} = T_i W_M \quad (6)$$

Execution time for *stop phase* of Pre-copy based live migration:

$$T_{stop} = \frac{W_S P}{B} \quad (7)$$

Total Migration Time of Pre-copy live migration

$= \sum_i^n T_i + T_{stop}$ (8)

In push phase of the proposed framework as shown in (1), the system first transfer the memory page except the Working Set list W_s that include the most recent pages during the migration time. In (5) for the push phase of the pre-copy based live migration, the system first transfers the all of the memory pages. The proposed framework reduces the transfer memory page using the working set list during the push phase of live migration. But the total migration of the proposed framework include not only the execution time for push phases and stop and copy phase but also the overhead for pre-processing phase that apply the working set prediction algorithm.

From the equations, we give a specific, numerical example to evaluate the total migration time of the proposed framework. Based on the experiment with Quake3 Server workload, we make the assumption that before migration. The virtual machine has the following states: M=131072 pages, B=5.625 MB/s, W_M=512 pages/s and the size of the page is 4KB. Then we have the results in Table 1. From these results, the proposed framework can migrate a virtual machine in less time than the pre-copy based live migration.

In Figure 4, we present the execution time for pre-processing phase that apply the working set prediction algorithm of the proposed framework with various working set size. According to the result, the larger size of working set may cause the more execution time.

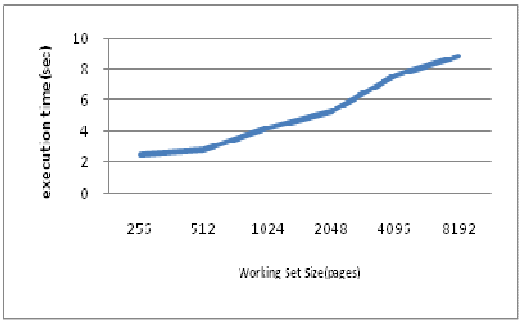

Figure 4. Execution Time for Pre-processing Phase of the Proposed Framework

In Figure 5, we present results comparing the execution time for push phase of the proposed framework with pre-processing phase and without pre-processing phase using various working set size. Although we need less execution time for push phase with pre-processing phase, the system has the overhead for pre-processing phase.

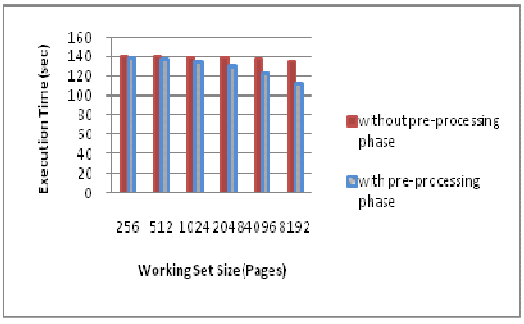

Figure 5. Execution Time for Push Phase of the Proposed Framework

In Figure 6, the calculation results show the total migration time of the proposed framework with various working set size. According to the calculation results, 32 MB (8192 pages) is suitable for Working Set list of the proposed framework. If the size of Working Set list increased, the execution time for pre-processing phase also increased. The system need to be balance the effect and overhead using pre-processing phase.

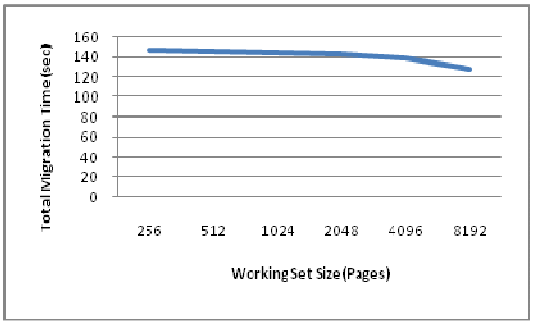

Figure 6. Total Migration Time for Proposed Framework

Figure 7 shows the comparison of total migration time for the proposed framework with pre-processing phase and pre-copy based live migration using various working set size. From these calculation results, we can reduce the total migration time in the proposed framework than pre-copy based live migration of virtual machine while the working set size is 1024 pages and above. In 256 pages of working set, the preprocessing phase can be overhead for total migration time.

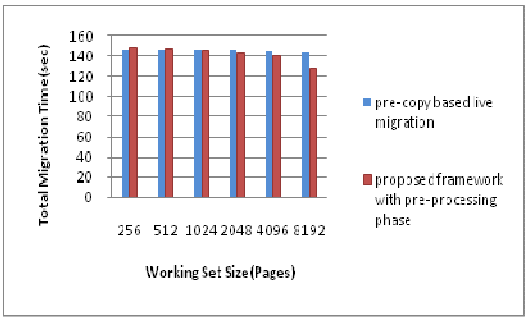

Figure 7. Total Migration Time of the proposed framework with pre-processing phase Vs pre-copy based live migration

5. Conclusion

In this paper, we have presented the framework which includes the pre-processing phase, push phase and stop phase for pre-copy based live migration. Pre-copy based live

migration ensures that keep downtime small by minimizing the amount of VM state. It provides to abort the migration should the destination VM ever crash during migration because the VM is still running at the source. It needs to reduce the total migration time and impact of migration on performance of source and destination VMs. We proposed the working set prediction algorithm which based on Least Recent Used (LRU) policy collects the most recent used memory pages as the Working Set list and collects the least recent used memory pages as the inactive list in pre-processing phase. The proposed framework migrate the inactive list in push phase and lastly it migrate the Working set list in stop phase to reduce the amount of transferred data and iteration times. From the calculation results, the proposed framework can reduce the total migration time in live migration of virtual machines.

6. References

[1] A. Kivity, Y. Kamay, and D. Laor, "kvm: the linux virtual machine monitor". *In Proc. of Ottawa Linux Symposium* (2007).

[2] 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.

[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] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim, "LRFU: A Spectrum of Policies that Subsumes the LRU and LFU Policies", IEEE Transactions on Computers, vol. 50, no. 12, pp.1352-1361, Dec. 2001.

[5] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the 18th International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009, pp.101–110.

[6] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan. Live virtual machine migration with adaptive memory compression. In Proceedings of the 2009 IEEE International Conference on Cluster Computing (Cluster 2009), 2009.

[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] 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

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

[10] 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.

[11] P. Lu and K. Shen, "Virtual machine memory access tracing with hypervisor exclusive cache". In ATC'07: *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*,pages 1–15, Berkeley, CA, USA, 2007. USENIX Association. ISBN 999-8888-77-6.

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

[13] 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.

[14] T. Yang, E. Berger, M. Hertz, S. Kaplan, and J. Moss. Automatic heap sizing: Taking real memory into account, 2004. URL citeseer.ist.psu.edu/article/yang04automatic.html.

[15] T. Yang, E. Berger, S. Kaplan, and J. Moss. CRAMM: virtual memory support for garbage-collected applications. In OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation, pages 103–116, Berkeley, CA, USA, 2006. USENIX Association. ISBN 1-931971-47-1.

[16] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High Performance Virtual Machine Migration with RDMA over Modern Interconnects", *Proceedings of IEEE Conference on Cluster Computing (Cluster 2007)*, Austin, Texas. September, 2007