Efficient Pre-Copy Live Migration of Virtual Machines for High Performance Computing in Cloud Computing Environments

Kasidit Chanchio
Department of Computer Science
Faculty of Science and Technology
Thammasat University, THAILAND
kasiditchanchio@gmail.com

Jumpol Yaothanee
Department of Computer Science
Faculty of Science and Technology
Thammasat University, THAILAND
lopmuj@gmail.com

***Abstract*—Pre-copy live migration is the most popular migration mechanism implemented in most hypervisors. However, it is not suitable for Virtual Machines (VMs) running computation-intensive and memory-intensive workloads because its operational behaviors depend on a configuration parameter, namely the maximum tolerable downtime. Although the default value of this parameter is good enough for normal web-based applications, it is too low for most HPC applications, which are computation-intensive and memory-intensive. Performing a migration with an inappropriate parameter may cause extensively long migration time or downtime. Defining this parameter is nontrivial since application workloads are generally unpredictable. This difficulty also makes it hard for cloud management systems to operate VM live migration automatically.**

In this paper, we propose the Memory-bound Pre-copy Live Migration (MPLM) mechanism that performs VM live migration without requiring the maximum tolerable downtime parameter. MPLM presents a new perspective of VM live migration, where the migration is performed based on the present state of VM computation without enforcing downtime constraints. During live migration, MPLM separates memory pages into dirty and non-dirty sets and transmits them in a multiplexing manner. MPLM has been implemented in a modified version of the QEMU-KVM software. Experiments have been conducted to evaluate MPLM performances against those of the traditional pre-copy mechanism. We performed a number of live migrations of VMs running four OpenMP NAS Parallel Benchmark programs. Experimental results show that MPLM is more efficient and easier to use than the pre-copy mechanism.

Keywords-cloud computing; virtualization; live migration

I. INTRODUCTION

Live migration is a useful mechanism for resource utilization in cloud computing environments. In a large-scale cloud computing system that consists of thousands of servers, hardware and software failures may occur on any server at anytime [1]. Live migration can be used to automatically evacuate VMs from partially malfunction servers to healthy servers. It can also be used to improve load balancing by relocating VMs from overloaded servers to lightly loaded ones.

Unfortunately, pre-copy live migration mechanism, the most popular mechanism implemented in most hypervisors, is not suitable for the large-scale cloud systems for two reasons. First, it is not designed for automatic live migration. The mechanism requires system administrators to manually supply a maximum tolerable downtime value to the migration algorithm before operate. In QEMU-KVM, the default value of this parameter is 300 milliseconds [2]. While this value may be suitable for VMs running simple web servers, it is inappropriate for VMs running computation-intensive and memory-intensive workloads [3]. Second, it is not trivial to find appropriate maximum tolerable downtime values for VMs running arbitrary workloads. In large-scale cloud systems, VMs are not only used to run applications, but they are also used for running nested VMs [4] and Guest-OS-level containers [5]. Therefore, their workloads are generally arbitrary and unpredictable. As a result, the maximum tolerable downtimes that are suitable for such VMs are hard to determine. We will show in this paper further that (1) the performances of the pre-copy mechanism vary greatly over different maximum tolerable downtimes, and (2) live migration performances are inefficient when the maximum tolerable downtimes do not agree with VM computation behaviors.

This paper presents the Memory-bound Pre-copy Live Migration (MPLM) mechanism that performs VM live migration within a memory-bound amount of time and does not require the maximum tolerable downtime. MPLM performs live migration in three stages: startup, live migration, and stop-and-copy stages. In the startup stage, MPLM initiates the tracking of dirty page generation in VM memory. In the live migration stage, MPLM classifies VM memory pages into dirty and non-dirty pages and transfers them in a multiplexing fashion to a destination VM. Once all the non-dirty pages are transmitted, MPLM enters the stop-and-copy stage, where it halts VM computation and transfers remaining dirty pages to the destination.

We have implemented MPLM in a modified version of QEMU-KVM 2.9.0 [6] and conducted a number of experiments to evaluate MPLM against the pre-copy mechanism. For simplicity, we refer to QEMU-KVM as KVM from now on since we run every VM with hardware-assisted virtualization enabled [7].
In our experiments, we perform live migration of VMs running four OpenMP programs from the NAS parallel benchmark suite [8].

From the experiments, we find the following results:

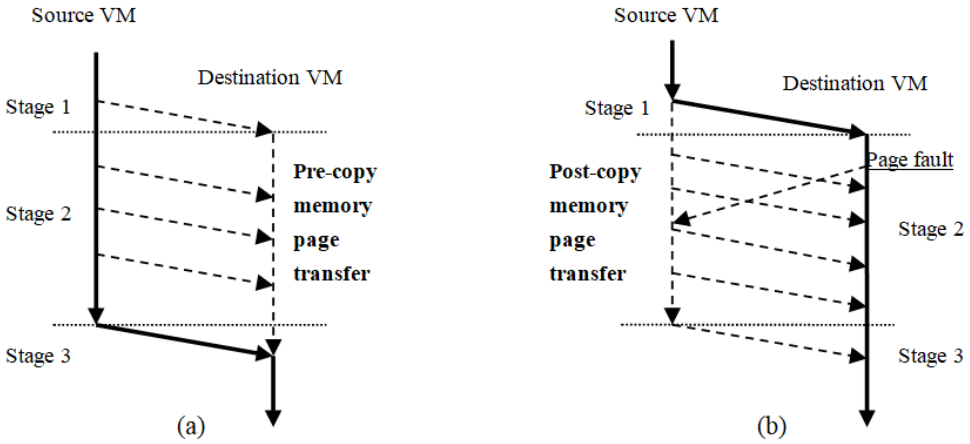

Figure 1. Diagrams show latency hiding scheme of (a) pre-copy and (b) post-copy migration.

First, MPLM is easier to use than the pre-copy because it does not require the maximum tolerable downtime.

Second, MPLM is more effective and efficient than the pre-copy. Unlike the pre-copy, MPLM does not produce extremely long migration time or downtime.

Finally, the downtimes of MPLM are correlated with the average dirty page generation rates during live migration stage. The higher the dirty page generation rate, the longer the downtime.

This paper is organized as follows. We discuss related works in Section 2. We describe the design and implementation of MPLM in section 3. Section 4 discusses experimental results. Finally, we give a conclusion and discuss future works in Section 5.

II. Related Works

VM live migration is an important tool to help manage resource utilization in cloud computing systems. Harchol-Balter and Downey [9] presented a theoretical study that shows the effectiveness of preemptive process migration, even when the memory transfer cost is high. We believe that their studies are applicable to VM live migration.

The designs of VM live migration mechanisms are based on the principle of latency hiding, where the migration algorithms will try to overlap VM memory transfer and VM computation as much as possible. There are mainly two kinds of VM live migration: pre-copy and post-copy migration.

The pre-copy mechanism was introduced in [10]. It operates in three stages: startup, live migration, and stop-and-copy (denoted Stages 1, 2, and 3 in Fig 1(a)). At Stage1, KVM marks every memory page as dirty page and instructs Linux kernel to track new dirty page generation.

It also creates a migration thread to handle memory pages transfer to a destination VM. Stage 2 performs latency hiding of VM memory page transfers by transferring the memory while the VM is running. The pre-copy mechanism performs the following steps:

1. The migration thread scans VM memory to find dirty pages and transmit them to the destination. The transmitted dirty pages are marked as non-dirty.
2. After the scanning reaches the last memory page, the migration thread calls the dirty page tracking mechanism to identify a set of new dirty pages, which have been modified by the VM during the memory scan.
3. If all new dirty pages can be transferred within the **maximum tolerable downtime**, the pre-copy mechanism will enter stage 3. Otherwise, it will perform stage 2 again.
4. At the stage 3, the mechanism stops VM computation, transmits remaining dirty pages, and resumes VM computation on the destination VM.

The major drawback of the pre-copy mechanism is the requirement of the maximum tolerable downtime. As mentioned earlier, it is difficult to find an appropriate value for this parameter and the default value is too low for VM running HPC workloads. Ibrahim et al. [3] proposed a mechanism to classify VM memory update patterns at runtime and generate appropriate ending conditions based on the detected pattern. Although their mechanism does not require manual configuration of the maximum tolerable downtime, some workloads may be too complex to classify and cause poor live migration performance.

Post-copy live migration also consists of three stages [11]. In Figure 1(b), the post-copy mechanism starts live migration by letting the source VM transfer a minimal subset of VM state to the destination VM so that it can immediately resume computation there (Stage 1). The mechanism halts the computation at the source VM. At Stage 2, while the destination VM is running, the post-copy mechanism instructs the source VM to transfer memory pages to the destination. Page faults may occur during VM computation on the destination if the requires memory pages have not yet been transmitted. To solve this problem, the destination VM temporarily halts the computation to retrieve the desired pages from the source. Finally, after every memory page on the source has been transmitted, the post-copy mechanism performs Stage 3 to end the migration and continue computation on the destination.

The disadvantage of the post-copy mechanism is its lack of reliability [12]. If the computation on the destination VM

crashes during Stage 2, although we can abort the migration and resume the computation on the source VM, works being done on the destination VM before the crash would still be lost. A complex checkpointing mechanism is needed to solve this problem [11]. On the contrary, this issue is not a problem for the pre-copy. In case the destination VM crashes during a migration, the pre-copy simply aborts the migration and resumes computation on the source VM.

III. MEMORY-BOUND PRE-COPY LIVE MIGRATION

MPLM is a new pre-copy live migration mechanism developed for two design goals. First, the mechanism must complete a migration within a bound period of time. Second it must be able to operate automatically without requiring manual configuration.

A. Design

Figure 2 shows the multi-threaded structure of KVM that consists of an I/O thread, vcpu-core threads, and a migration thread. The I/O thread is the main thread of KVM that is responsible for interacting with users, performing I/O operations, and orchestrate management operations such as live migration. The vcpu-core threads are used to execute workloads on virtual cpu cores of a VM. KVM assigns one thread per core. Finally, the migration thread is spawn to perform live migration operations.

The operations of MPLM can be described in three stages: startup, live migration, and stop-and-copy. In the startup stage (denoted S1 in the figure), the I/O thread makes a TCP connection with a destination hypervisor, initializes necessary data structures for migration operations, and call $Bitmap\_sync(0)$ function in the figure. This function instructs KVM kernel module to start keeping track of dirty page generation. MPLM also creates dirty set and non-dirty set data structures to store ids of dirty pages and non-dirty pages, respectively. The dirty set is initially an empty set, whereas the non-dirty set contains every memory page.

In the live migration stage (S2) MPLM creates a migration thread to perform live migration concurrently with the computation of the VM. The migration thread operates in a series of epochs. Each epoch is an interval of at most *Inv* seconds as illustrated in Figure 2. We call this *Inv* period the MPLM interval and set it to 3 seconds by default. The $Bitmap\_sync(i)$ function, where $i \geq 0$, marks the beginning of every epoch. At Epoch 0, MPLM only transmits non-dirty pages to the destination because the dirty set is empty. Throughout MPLM data transfer, the transmitted pages from either set are immediately deleted from that set. At the beginning of Epoch 1, the $Bitmap\_sync(1)$ function halts VM computation shortly to retrieve ids of new dirty pages the VM generated during the last epoch. These page ids are added to the dirty set and excluded from the non-dirty set. At the end of $Bitmap\_sync(1)$, MPLM resumes VM computation and lets the migration thread transfer pages from the dirty and non-dirty sets in a multiplexing fashion (see more details in the next section). After the *Inv* time period is over, the $Bitmap\_sync(2)$ function is called to obtain a new set of dirty pages. Again, MPLM adds them to the dirty set and excludes them from the non-dirty set. It is worth noting that the non-dirty set size reduces as stage S2 progress. MPLM repeats the data transfer and the calling of $Bitmap\_sync(i)$ until the non-dirty set is empty, the **ending condition** of the live migration stage. In Figure 2, the ending condition is satisfied after the invocation of $Bitmap\_sync(3)$.

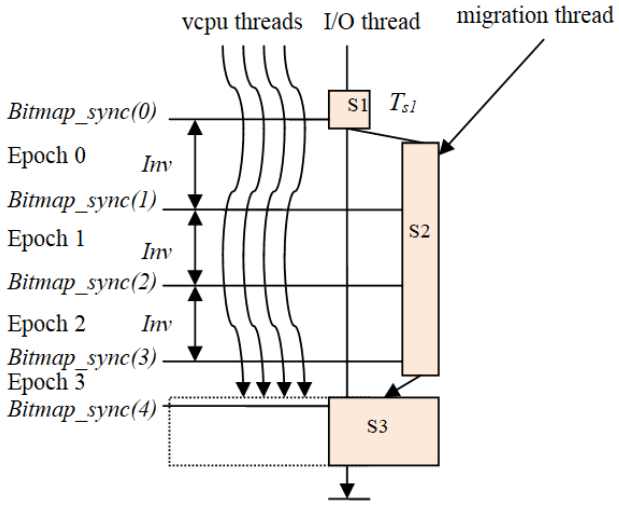

Figure 2. The structure and operations of MPLM

Finally at the stop-and-copy stage (S3), MPLM halts VM computation on the source computer, invokes $Bitmap\_sync(4)$ for the last time to update the dirty page set, and transfers every remaining dirty page to destination. Once the transmission completes, MPLM resumes VM computation on the destination host.

B. Data Transmission

The algorithm for data transmission during the live migration stage transfers memory page to the destination VM in a multiplexing manner. The algorithm relies on two pointer variables: the dirty-page and non-dirty-page pointers. At the beginning, both pointers point to page 0. At Epoch 0, MPLM scans memory and transmits only non-dirty pages. MPLM uses the non-dirty-page pointer for these operations. If the non-dirty-page pointer points to a non-dirty page, it transmits that page and increases the pointer value by one. Otherwise, it simply increases the pointer by one. At the end of Epoch 0, the non-dirty-page pointer would point to some page in memory while the dirty-page pointer still points to page 0.

At Epoch 1 and later, MPLM continues the memory scan and transfers both dirty and non-dirty pages. During an Epoch, MPLM repeatedly scans and transmits batches of memory pages. Each batch contains 100 pages. In each batch, MPLM scans and transfers dirty pages for 50 times and then turn to scan and transfer non-dirty pages for another 50 times. The scanning and transfers of dirty pages are similar to that of the non-dirty pages except we use the dirty-page pointer instead of the non-dirty-page pointer. The scanning and transfers of dirty and non-dirty pages are performed as described above from Epoch 1 on until the ending condition is satisfied.

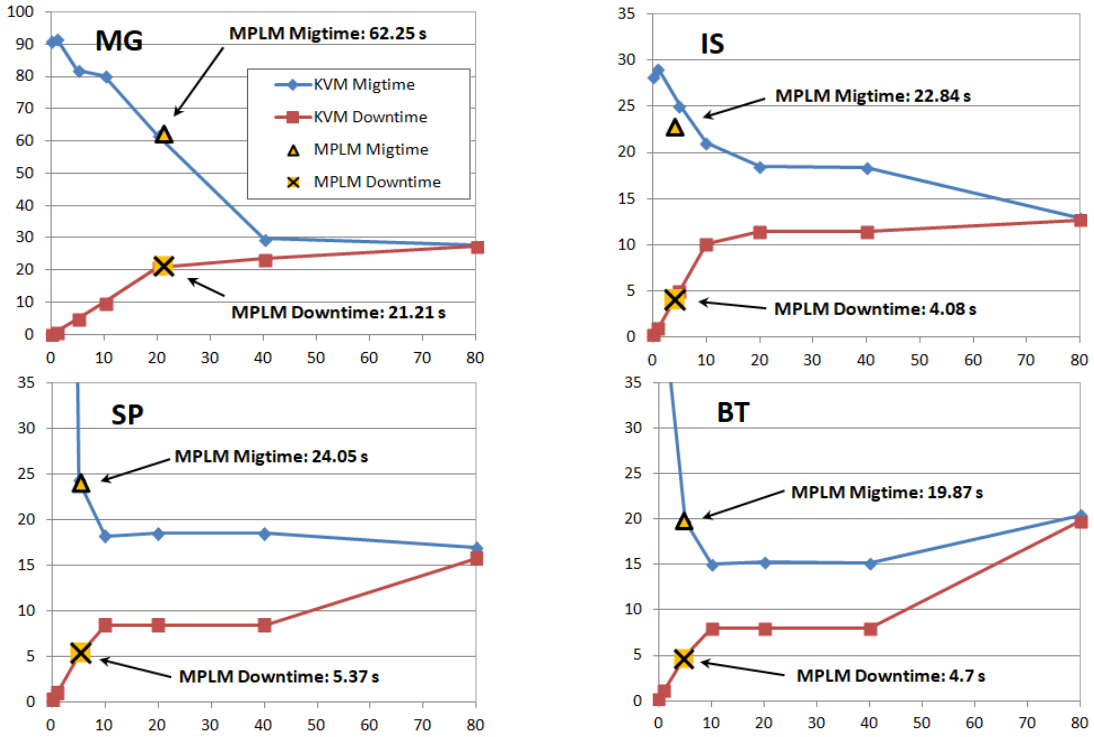

Figure 4. The total migration time and downtime performances of MPLM and those of the pre-copy mechanism.

IV. EXPERIMENTS

We have conducted a number of experiments to evaluate MPLM performance against those of the pre-copy mechanism and to study MPLM downtime. In our experimental setup, we use three host computers: the controller, source, and destination hosts. The controller host is responsible for orchestrating the experiments. The source and destination hosts are the source and destination of migration. We run an NFS server on the destination host and let the source host be an NFS client. The NFS network is separate from the network used for live migration. Both the source and destination hosts are a HP Proliant server with a 12 core AMD Opteron 2.1 GHz CPU, 48 GB Ram, and a 900GB 15K RPM SAS hard drive. They both run Ubuntu 14.04 and are connected by a 1 Gbps network.

We have installed the MPLM enabled KVM and the original KVM 2.9.0 on the source and destination hosts. In the experiments, the controller starts each run by invoking the hypervisor on the destination host to wait for data transfer. After that, it runs a source VM and executes a program on it. Finally, the controller instructs the hypervisor of the source VM to perform a migration. In every migration, the **maximum data transfer rate limit** is set to 1 Gbps.

Each VM is configured to have 8 vcpus and 8 GB of Ram, and run Scientific Linux 6.5. On each experimental run, the VM execute the OpenMP version of one of the MG, IS, SP, and BT class C programs of the NPB Benchmark [8]. After the program started, the controller wait until the memory usage of each program reaches its maximum working set size before instructing the VM to migrate.

A. *Performance compparisons of MPLM and the Pre-copy*

Figure 4 shows the migration time and downtime performances of MPLM and those of the pre-copy mechanisms for the migration of the MG, IS, SP, and BT benchmarks. For the pre-copy mechanism, the x-axis represents the maximum tolerable downtime. On the other hand, for MPLM, the x-axis represents the actual downtime. The y-axis shows total migration time and actual downtime of both mechanisms in seconds. Every data point is an average of 10 runs.

From the figure, the migration times and downtimes of MPLM are presented as coordinates while those of the pre-copy are presented by graph lines. The label **MPLM Migtime** denotes the total migration time of MPLM and the label **MPLM Downtime** denotes the actual downtime. For the pre-copy, the total migration time performances of each benchmark are illustrated by the **KVM Migtime** graph, while the downtime performances are presented by the **KVM Downtime** graph. Each pre-copy graph connects a series of timing results when the maximum tolerable downtimes are set to 0.3 (the default value of KVM), 1, 5, 10, 30, 60, 80 and 120 seconds, respectively.

From the results, the migration time and downtime performances of MPLM are presented as coordinates rather than graph lines (of the pre-copy) because MPLM eliminates the need for the maximum tolerable downtime parameter. Thus, MPLM is easier to use than the pre-copy mechanism.

By considering the **KVM Migtime** and **KVM Downtime** graphs, the total migration times and downtimes of the pre-copy mechanism change considerably as the maximum tolerable downtime increases. On the **KVM**

Migtime graph of every benchmark, when the maximum tolerable downtime is 0.3 second, the default value of KVM, the total migration times become extremely high. For examples, the total migration times of the SP and BT benchmarks using the default maximum tolerable downtime are 984.41 and 622.62 seconds, respectively. They are in fact higher than the application execution times on every benchmark. When the maximum tolerable downtime increases, the total migration times reduce and converge to some values. On the other hand, the downtime performances of the pre-copy are very low when the maximum downtime is 0.3 second. They increase sharply as the maximum tolerable downtime increase and converge to some values as shown in Figure 2.

The pre-copy mechanism produces the performance above because every benchmark is computation-intensive and memory-intensive. Enforcing the maximum tolerable downtime requirement of 0.3 second is unsuitable with the amount of dirty pages generated by the applications. As a result, pre-copy mechanism extends its live migration stage until the benchmark finishes. This kind of migration behavior is ineffective and inefficient. It wastes a lot of migration resources and bandwidths. However, if we choose the maximum tolerable downtime too high (e.g. 40 or 80 seconds), the downtimes become too high as well. In Figure 2, when the maximum tolerable downtime is 80 seconds, the downtime is more or less equal to the total migration on every benchmark. This behavior is inefficient because there is only a little overlap (or no overlap at all) of VM computation and VM data transfers.

MPLM provides a middle way. It automatically generates efficient migration time and downtime performances for every benchmark. In Figure 4, **MPLM Migtime** and **MPLM Downtime** coordinates are close to the low ends of the **KVM Migtime** and **KVM Downtime** graph lines for the IS, SP, and BT benchmarks. In case of the MG, **MPLM Migtime** and **MPLM Downtime** coordinates locate near the elbow point of the **KVM Migtime** and **KVM Downtime** graph. According to MPLM algorithm, MPLM performances reflect the actual state of VM execution during live migration. MPLM automatically produces the total migration time and downtime performances which would allow the overlapping of VM computation and VM state transfer as much as possible.

B. MPLM Downtime and Dirty page generation

From the experiments, we noticed that MPLM downtimes are generally correlated with the dirty page generation rate of the VM during live migration. From Table 1, the higher the average dirty page generation rate during the live migration stage of MPLM, the higher the downtime. MPLM live migration of the three benchmarks in Table 1 finishes before the executions of the benchmarks finish. The IS performance is excluded from the Table because the migration behavior of the benchmark is different. The execution of the IS benchmark finishes before the completions of live migration.

TABLE I. THE CORRELATION BETWEEN MPLM DOWNTIMES AND AVERAGE DIRTY PAGE GENERATION RATES.

Benchmark	***MPLM Downtime (sec)***	***Dirty Page Generation rate (pages/sec)***
BT	4.70	35061.61
SP	5.37	47370.42
MG	21.21	207636.52

V. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a novel pre-copy live migration mechanism called MPLM. Unlike the traditional pre-copy mechanism, MPLM does not require users to manually configure the maximum tolerable downtime parameter, which is difficult to define for VMs running computation-intensive and memory-intensive workloads. This property makes MPLM easy to use and suitable for datacenters that want to perform live migration automatically. MPLM operates live migration within a memory-bound length of time. It is more efficient than the pre-copy mechanism as shown in the experiments. Source codes and other information about MPLM are available at [13].

In the future, we plan to evaluate MPLM further using different kinds of VM workloads. In the long-run, we plan to integrate MPLM into a cloud management system to perform VM live migration automatically and evaluate its impacts on the overall application execution and resource utilization performances.

REFERENCES

[1] Al Geist, "How To Kill A Supercomputer: Dirty Power, Cosmic Rays, and Bad Solder", IEEE Spectrum, March 2017

[2] https://download.qemu.org/qemu-2.9.0.tar.xz

[3] K.Z. Ibrahim et al. "Optimized pre-copy live migration for memory intensive applications," *Supercomputing (SC),* 2011

[4] Muli Ben-Yehuda et al. "The Turtles Project: Design and Implementation of Nested Virtualization," *9th USENIX OSDI*, 2010

[5] https://www.datamation.com/open-source/linux-containers-vs-virtual-machines.html

[6] https://wiki.qemu.org/ChangeLog/2.9

[7] A. Kivity et al. "kvm: the linux virtual machine monitor," *Proc. of Linux Symposium*, 2007.

[8] NPB 3.3, http://www.nas.nasa.gov/publications/npb.html#url

[9] Mor Harchol-Balter and Allen Downey. "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," ACM Transactions on Computer Systems vol. 15, no. 3, August 1997, pp. 253-285

[10] C. Clark et al. "Live migration of virtual machines," *USENIX NSDI*, 2005

[11] M. R. Hines et al. Post-copy live migration of virtual machines. SIGOPS Oper. Syst. Rev. 43, 3, July 2009

[12] https://github.com/qemu/qemu/blob/master/docs/migration.txt

[13] https://github.com/kasidit/qemu-mplm