Efficient Pre-Copy Live Migration with Memory Compaction and Adaptive VM Downtime Control

Guangyong Piao, Youngsup Oh, Baegjae Sung, and Chanik Park
Department of Computer Science and Engineering
Pohang University of Science and Engineering (POSTECH)
Pohang, South Korea
{tmipyiong, youngsup, jays, cipark}@postech.ac.kr

***Abstract*— Virtual machine (VM) live migration is an important feature of the virtualization technique. Pre-copy method is typically used to support live migration in most virtualization environments. It is observed that pre-copy may not work in some situation where memory-intensive applications are running in KVM [3]. Moreover, pre-copy live migration does not consider high duplication ratio between memory and root file system as well as between memory snapshots. This paper proposes two techniques to improve pre-copy live migrations: 1) memory compaction technique based on disk cache and memory snapshot; 2) adaptive downtime control scheme based on the history of VM's memory update information called WWS (Writable Working Set). We have successfully implemented the proposed method in KVM. It is shown that the proposed method can support live migration under any circumstances with little overhead.**

Keywords: VM Live Migration, Downtime Control, Memory Compaction, KVM

I. INTRODUCTION

Virtual machine (VM) live migration is an essential feature of virtualization technique. With this feature VMs can be migrated lively between different hosts, without any service disruption to support load balancing and fault tolerance [1, 2]. Pre-copy is the popularly adopted live migration technique in most virtualization environments like Kernel-based Virtual Machine (KVM) or Xen.

It is observed that there are two significant weaknesses of pre-copy live migration techniques. First, pre-copy live migration requires all of the memory pages of a source VM should be sent to a destination VM. We need to consider how to reduce the network traffic, based on deduplicated information [5, 7]. Depending on VM's characteristics, there may be some opportunities for additional deduplication information. Second, pre-copy live migration should iteratively send all memory pages updated during the previous memory page transfer phase. If memory-intensive applications are running, then there is a high chance for pre-copy live migration to continuously send the newly updated memory pages forever without termination [3].

In this paper, two techniques are proposed to make up the weaknesses of pre-copy live migration technique: 1) memory compaction technique based on disk cache and memory snapshot; 2) adaptive downtime control scheme based on the history of VM's memory update information called WWS (Writable Working Set).

The rest of the paper is organized as follows. Section 2 introduces background of pre-copy live migration technique, the problem in KVM live migration method as well as related work. Section 3 explains the detailed design of our proposed migration method. Section 4 shows the effectiveness of the proposed migration method by presenting the experimental results. Finally, Section 5 gives a brief summary of this paper.

II. BACKGROUND

A. Pre-copy Live Migration

Pre-copy is a popular VM live migration technique, which has been implemented on many kinds of hypervisors, such as KVM, VMware, and Xen. The process of pre-copy live migration can be divided into three phases, and the detailed description of each phase is as follows.

1) Transfer entire memory pages

Before starting transfer of all memory pages, the live migration process first set the entire memory pages as read-only, to detect the pages updated, e.g. dirty, during the first phase of live migration. The information on the updated pages is maintained as a bitmap called dirty bitmap by page fault handler. Then the migration process starts transferring all memory pages of a source VM to a destination host.

2) Iteratively transfer newly dirty pages

In this phase, the dirty pages identified by the dirty bitmap are to be transferred to the destination. Before starting transfer, dirty bitmap should be cleared and we assume that all the memory pages are set to read-only again. At the end of each phase, the migration process tries to computes the expected transfer time, that is, VM downtime, required to transfer all the dirty pages in the dirty bitmap. If the expected down time becomes shorter than the pre-configured downtime threshold called VM max downtime, then the migration process switches to the final stop-and-copy phase. Otherwise, the migration process repeats the phase 2 iteratively with the current dirty bimap.

3) Stop and Copy

The migration process suspends the execution of the source VM, transfers all the dirty pages identified in the dirty bitmap, and finally resume the VM on the destination side.

978-1-4799-6719-3/14 $31.00 © 2014 IEEE
DOI 10.1109/BDCloud.2014.57

B. Weakness of Pre-copy Live Migration

The most of the live migration time in pre-copy scheme is spent on Phase 1 or Phase 2, and the portions of the migration time spends on each phase are highly depend on the behavior of a source VM. If the memory of the VM does not change too much during the migration, a great portion of the migration time would be spent on Phase 1. Otherwise, the migration time would be spent mostly on Phase 2.

In order to optimize the operation of Phase 1, we need to consider which resources in the destination host may include the same or similar information with the memory pages of a source VM. However, the pre-copy of KVM tries to send every non-zero memory pages to the destination side, without considering any compression or deduplication technique.

Another significant weakness of pre-copy live migration is in the phase 2. If the number of dirty pages obtained from the dirty bitmap is large, the migration process may not switch to the final stop-and-copy phase. This means that pre-copy live migration may not complete the mission, transferring dirty memory pages endlessly. This is shown in previous research [3, 4] that KVM live migration could not cope with memory intensive cases. This is due to the fact that pre-copy live migration is based on the statically configured downtime threshold, VM max downtime. A simple solution is to set the downtime threshold to a very high value, but we need a more sophisticated technique.

C. Related Work

To reduce the size of transferring data via network, Jo et al. [5] proposed an efficient live migration method by maintaining the mapping information, between the memory page frame number (PFN) and the disk logical block address (LBA), for the page cache. If a memory page belongs to the page cache, only the mapping information is sent to the destination host, and based on that information the memory page would be filled directly from network attached storage (NAS). However, their approach could not work well when most of the memory spaces are possessed by processes, because page caches are only available when there is enough free memory space.

Determine the time point of force stop and copy, when the live migration is never ended up, is another important research issue. Khaled et al. [3] analyzed the patterns of the High Performance Computing (HPC) applications and classify those patterns into three types, designed an algorithm to detect the never-ended-up circumstance. After a successful detection, the migration phase would be switched to phase 3 by a force. However, they did not concern about which time point is most appropriate to minimize the migration downtime.

III. Efficient Pre-Copy Live Migration

A. Memory Compaction

1) Duplication between Memory and Disk

Modern operating systems cache recently accessed disk data or read-only data blocks on the memory, to reduce the latency of disk access [5, 6]. Besides, page prefetching techniques which implemented for improve the interactive performance, also additionally fetch more disk blocks on the memory [7].

Because these data blocks are aligned by page size (4KB) on the memory, if we can figure out the frequently used data blocks, and statically maintain these data blocks on both sides, a mapping information between the memory page-frame number (PFN) and the offset of the duplicated data blocks inside the set of data blocks, is enough for transferring a memory page which is duplicated with the static disk cache. Therefore the transferring data size for those duplicated blocks can be reduced from 4KB to 16Byte (64bit address, 64bit offset).

2) Duplication between Current and Previous Memory Snapshots

Previous memory snapshot of a VM is available in many cloud service environment. Especially, in desktop as [8] a service (DaaS) model, a VM migrates between the cloud server and user's local machine frequently. When a migration just ended up, an identical memory snapshots would be remained on both sides.

When the VM need to be migrated back to the original source, after runs for a long time, with a previous snapshots, the memory contents of the VM could be classified into three categories, based on the relationship between the current memory page and the previous memory snapshot: 1) identical; 2) similar; 3) different. For page type 1, only the PFN need to be transferred to the destination side. Besides, for page type 2, the memory pages can be reconstructed on destination side with the PFN and diff information. XBZRLE [9] is an efficient delta compression method which is suitable for memory live migration, since we adopt the XBZRLE to detect the similar memory pages and delta extraction. Finally, only if the memory page belongs to the type 3, the page transfers to the destination as pure text.

3) Implementing Memory Compaction

The description of each component in proposed architecture is as follows: (Fig. 1)

Disk Cache: In order to rebuild the memory with page-level data deduplication with disk, an identical set of disk blocks should be maintained on both sides. However, maintaining a large scale disk is a significant overhead. The data blocks stored in the disk cache mainly consist of frequently used shared libraries or binary files. This disk cache does detect the duplication information between memory pages and typically shared files in the root file system of a destination host.

Disk Cache Tree: A SHA-1 hash value store of disk cache. The offset of a data block in disk cache would be retrieved with the hash value of transferring memory page.

Memory Snapshot: Memory snapshot of the VM, generated at the end of the last successful migration from the current destination to the current source. It can be used to perform page-level data deduplication as well as XBZRLE delta encoding with current transferring memory page.

Memory Snapshot Tree: SHA-1 hash value store of memory snapshot. It is similar with the cache tree.

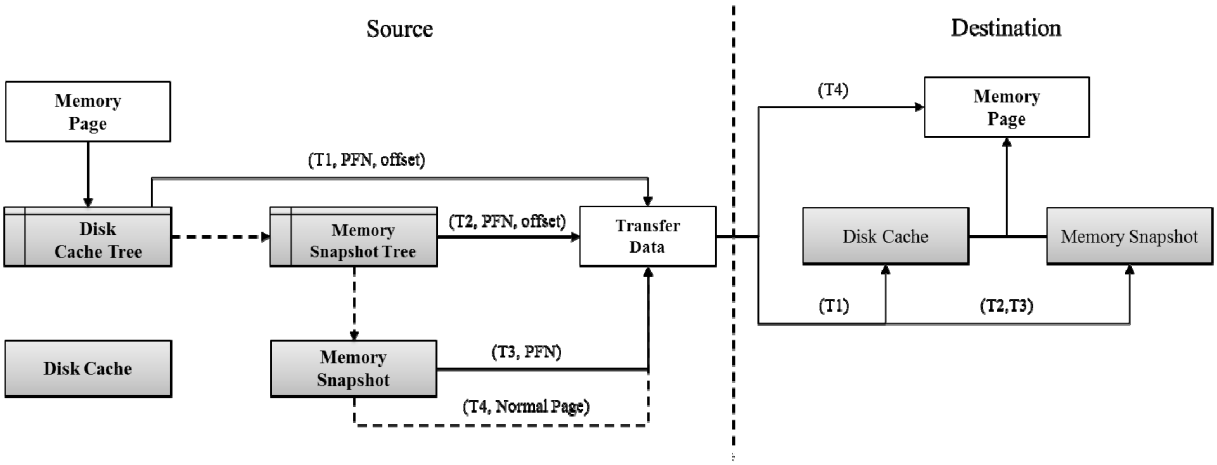

Figure1. The Architecture of Memory Compaction Technique: Deduplication based on Disk Cache and Memory Snapshot. (Arrow: the hash tree lookup or XBZRALE encoding is succeed, Virtual Arrow: the hash tree lookup or XBZRLE encoding is failed)

Type Header: One-byte-header which identifies the type of transferring data.

At the first time when a VM migrates to the destination side, a disk cache can be utilized to reduce the transferring memory size in phase 1. In this case, the migration process calculates the SHA-1 hash value of each transferring memory, and check whether the hash value exists in the cache hash tree. If the hash value exists in the cache hash tree, a PFN and offset of the duplicated block transfers to the destination side, and that memory can be directly constructed from the disk cache on destination side (T1 in Fig. 1). After the first migration ended up, an identical memory snapshot could be generated on both sides. From that point the return back migration can be additionally accelerated by utilizing the memory snapshot. In this case, when there is no matched hash value of memory block after looking up the cache hash tree, the migration process traverses the snapshot tree to detect page-level duplicated data with memory snapshot (T2 in Fig. 1). Similar with the disk cache case, the PFN and offset of the data blocks in snapshot is enough for reconstructing the memory on destination side. Even there is no duplicated page-level data blocks in memory snapshot, the XBZRLE delta encoding with the memory page with the same PFN in memory snapshot, would figure out sub-page level duplicated memory contents (T3 in Fig. 1). Finally, if all of the above attempts were failed, we transfer the page with no modification (T4 in Fig. 1).

In our implementation, a memory size amount of heap region is allocated, and when each memory page transfers to the destination side, the migration process fill up the heap with the memory contents based on the PFN information. The constructed snapshots write to the disk, at the end of a successful live migration.

B. Adaptive VM Downtime Control

1) Analysis of Memory Intensive Workload

VM migration could not be terminated when the memory updating rate is higher than the data transferring rate. The main reason is that of the large expected downtime, which is related with the WWS generated in that round, always exceeds the predefined max downtime. To analyze the pattern of WWS of memory intensive workload, we ran Canneal and X264 workload belong to The Princeton Application Repository for Shared-Memory Computers (PARSEC) on Ubuntu12.04 64bit and 1Gbps NIC, and monitored the expected downtime at the end of each iteration. Two kinds of similar patterns are observed in most of non-migrated workloads, include memory intensive workload in PARSEC and high resolution movie workloads (Fig. 2). In the first pattern, a large amount of pages became dirty in each round, with a stable size (from the 3rd to the 40th iteration in Canneal, and from 3rd to the 20th iteration). This phenomenon is due to the working set of the application does not change too much during the execution. Besides, when a new memory intensive process was just started or finished input initialization or ended up, the expected downtime would become a peak value, since the WWS dropped or increased rapidly.

2) Extract the Trend of WWS with Linear Least Square Regression

The linear regression is used to determine the direction of a trend. In this case, we adopted the linear least square regression method to observe whether the WWS pattern

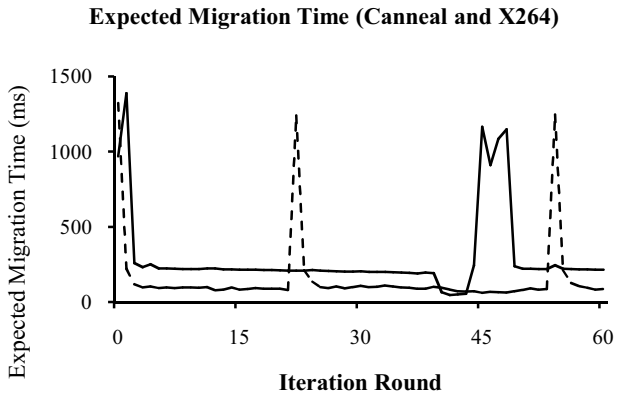

Figure 2. The Expected Migration Time (ms) after each Iteration. (Line: Canneal, Virtual Line: X264)

could be figured out by the Equation 1 and Equation 2. In general case the migration ended up with in 5 iterations. So if the migration does not finish until the 5th iteration, we can use previous 4 histories as sample points of WWS, and calculate the linear regression function based on those sample values. Therefore we defined the number 5 as a window size, which means we only consider the latest 5 histories of WWS (in MB) include the WWS generated in this round.

We observed that with linear regression: 1) a sharp peak would be converted to a smooth curve; 2) when the WWS does not change too much, the value of *a* in linear regression function is always between -10 to +10 (Fig. 3).

$$y = ax + b \quad (1)$$

$$a = \frac{n\sum(xy) - (\sum x)(\sum y)}{n\sum x^2 - (\sum x)^2} \quad (2)$$

Equation 1. The Equation of Linear Least Square Regression. (x: explanatory variable, y: dependent variable, n: the number of samples, window size)

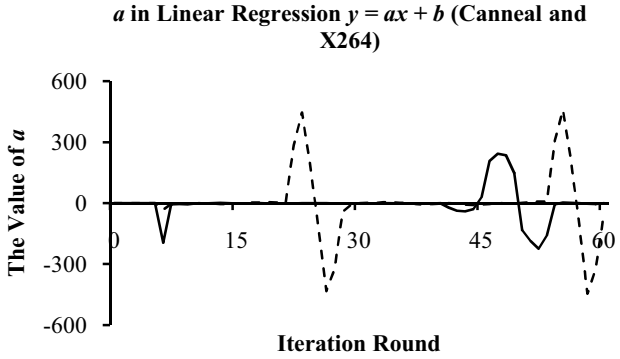

Figure 3. *a* in Linear Least Square Regression with Window Size 5. (Line: Canneal, Virtual Line: X264)

3) Adaptive VM Downtime Control Algorithm

With the observations in 4.2, the memory pattern can be classified into two types: stable state where the value of *a* is between -10 to +10, and unstable state where the value of *a* is larger than +10 or less than -10. When the dirty page size is stable and could not finish the migration, the max downtime should be lazily increased to a larger value to exceed the expected downtime. So the expected downtime would be increased based on the gap between expected downtime and max downtime. Otherwise if the size of WWS changes with unstable pattern, the change of max downtime follows the value of a (Eq. 1). When the dirty page size increasing in very low rate (less than a MB/s), and the gap of expected downtime and max downtime is less than that low rate, the max downtime could not excess the expected downtime. In such a case, the algorithm choose a larger one between two values (line 10 in Algorithm. 1). In this algorithm most of the CPU time of would be consumed by computing the equation of linear regression, so the time complexity is O (n^2).

Algorithm1. Adaptive VM Downtime Control Algorithm

```
1:  procedure CALC_MAX_DOWNTIME (bandwidth: In, iteration_round InOut, dirty_page_size: In, max_downtime InOut, expected_downtime In)
2:  Insert dirty_page_size into history_array
3:  if iteration_round > window_size then
4:      a ← linear_leat_square_regression(history_array)
5:  end if
6:  if -10 < -a < 10 then
7:      if previous_state is stable then
8:          max_downtime ← max_downtime + delta
9:      else
10:         delta ← Max[(expected_downtime -
                max_downtime)/window_size, 2a/bandwidth]
11:         max_downtime ← max_downtime + delta
12:     previous_state ← stable
13: else
14:     max_downtime ← max_downtime + a/bandwidth
15:     if max_downtime < 20 then
16:         max_downtime ← 20
17:     end if
18:   previous_sate ← unstable
19: end if
20: iteration_round ++
21: end procedure
```

IV. Evaluation

A. Experimental Environment

The two migration host have 12 Intel Xeon E5-2530 CPUs, 4TB disk and 48GB of RAM, ran on Ubuntu server 12.04 64Bit. The hypervisor of each machine was a modified version of QEMU-KVM 1.2.0 which had been implemented the proposed efficient live migration technique. Two kinds of workloads, desktop and server workload, were adopted in the experiments. The desktop workload consists of libreoffice and firefox operations, and the server workload consists of write intensive workload in PARSEC. The network configuration is different from desktop and server workload, in order to emulate a client to server and server to server migration environment. We adopted 100Mbps network for desktop workload, and 1Gbps network for server workload. Additionally the results were the average values of three identical experiments. To recover the same environment of VMs, we captured the snapshot of the VM on the qcow2 image just before a migration starts.

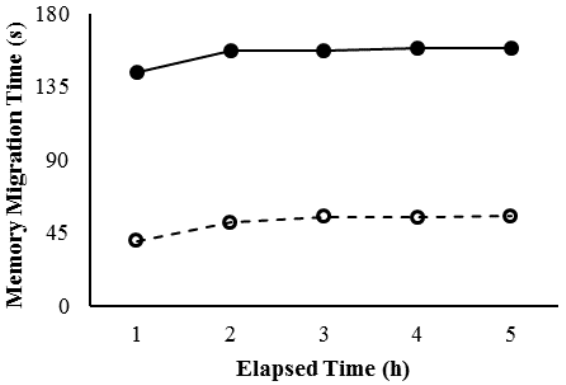

Figure 4. The Memory Migration Time of a source VM: x- axis represents how long the source VM has been running before a memory snapshot is taken (Line: Original KVM, Virtual Line: Memory Compaction Technique)

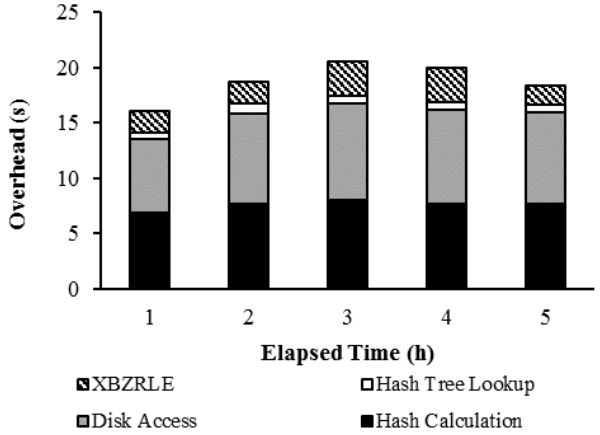

Figure 5. Details of Reduced Traffic Size (MB) by Proposed Memory Compaction: x-axis represents how long a source VM has been running before a memory snapshot is taken

B. *Effects of Memory Compaction Techniques*

In the first experiment, we migrated a VM from a source host to the destination host after running the desktop workload for 1 hour. After the migration, identical snapshots would be generated on both source and destination sides. We rolled back the VM to the state before the migration, and ran the desktop workload for another 5 hours, and took snapshots in every hour. Then the VM rolled back to the previous moments and migrated the VM to the destination. In this case, a previous memory snapshot of the first hour can be utilized to reduce the transferring memory size by page level data deduplication and sub-page level XBZRLE delta encoding. In the original KVM the total migration time increased, since the increase of non-zero memory page size and dirty disk block size (Fig. 4). In the proposed method, the time spent on transferring memory pages are reduced significantly. This is because the reduced time by transferring less data is much larger than the overhead of our migration method.

The reduced traffic is mainly due to the deduplication with previous memory snapshot. Besides, the size of duplicated data with disk cache was stable in five cases, since the disk cache is not evicted when there is enough free memory space. Additionally, the average compression ratio by XBZRLE delta encoding is around 2 (Fig. 5). In terms of overhead, both of SHA-1 hash calculation time and disk access time possessed a great portion. The SHA-1 overhead increases linearly with the increase of non-zero pages. In our experiments, the SHA-1 computing time for 2GB memory is around 8second, which means that 96MB of benefit in traffic size would hide this overhead. The disk access overhead are consists of the memory snapshot access time on source side for XBZRLE and memory snapshot access time and disk cache access time on destination side. So on the source side, the disk access overhead is related with the number of pages which are not duplicated with both disk cache and memory snapshots. On the destination side, a large degree of the disk access overhead means more memory pages are constructed from the disk instead of transferring via network (Fig. 6).

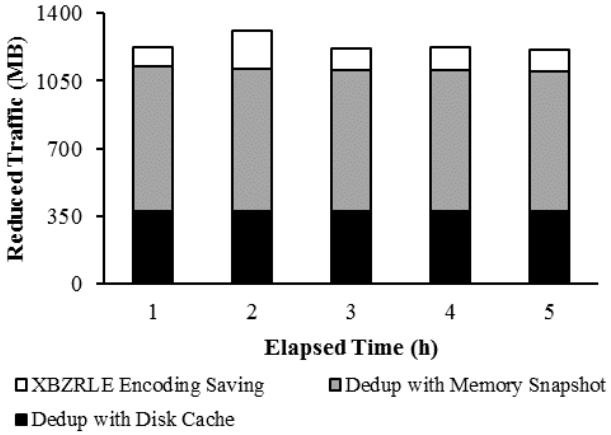

Figure 6. Overheads analysis of the Proposed Method: x- axis represents how many hours the source VM has been running before taking a memory snapshot.

C. *Effects of Adaptive Downtime Control Technique*

To observe the efficiency of the WWS history based adaptive downtime control method, we select the Canneal workload in PARSEC, and graph analytics workload from Cloud Suite. To make the workload more intensive, in the first experiment we ran a Canneal after freshly booted-up and started the live migration. While migrating, we activated another Canneal workload in the 4th iteration in phase2. After fifth iteration, the max downtime started to raise up, because the downtime control algorithm detected the migration would not be ended up. Even there was a peak point at 7th iteration, the increase of max downtime was not influenced by this special points, and successfully migrated within a reasonable downtime (Fig. 7).

In the next experiment, we ran a Canneal after freshly booted-up and started another Canneal at 6th iteration, similar with the previous experiment. The difference is that, we stopped both of Canneals at 10th iteration. With the adaptive downtime control method, the migration process successfully detected the trend of the WWS, and finished the migration with a very small downtime (Fig. 8).

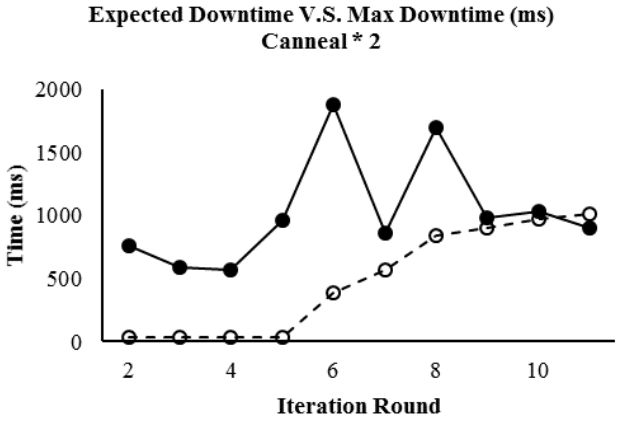

Figure 7. Expected Downtime V.S. Max Downtime in case of Cannel *2 workload, with Adaptive VM Downtime Control (–●–: Expected Downtime – –○– –: Max Downtime)

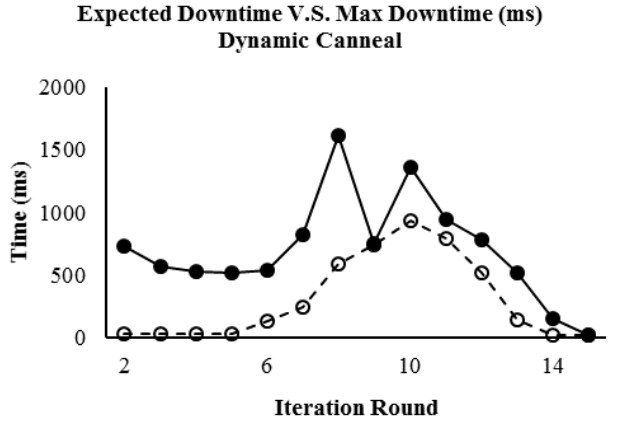

Figure 8. Expected Downtime V.S. Max Downtime in case of Dynamic Cannel workload, with Adaptive VM Downtime Control (–●–: Expected Downtime – –○– –: Max Downtime)

In the last experiment, we ran a graph analysis workload on a freshly booted-up VM. Even the size of WWS changes unstably, our method also could finish the migration (Fig. 9).

V. CONCLUSION

In this paper, we analyzed two critical weaknesses of the KVM pre-copy live migration technique. To make up these problems, we proposed a memory compaction technique based on disk-memory deduplication cache and memory snapshot, as well as an adaptive VM downtime control technique. We have implemented these techniques in KVM. It is shown that the memory compaction techniques could reduce memory transfer time by a factor of 2, mostly in the first phase. The experimental results also proved that the adaptive VM downtime control technique successfully handled the live migration for the VM running memory intensive workloads. However, the memory compaction technique might not work well when the memory contents of target migrating VM change too much (e.g. the VM running

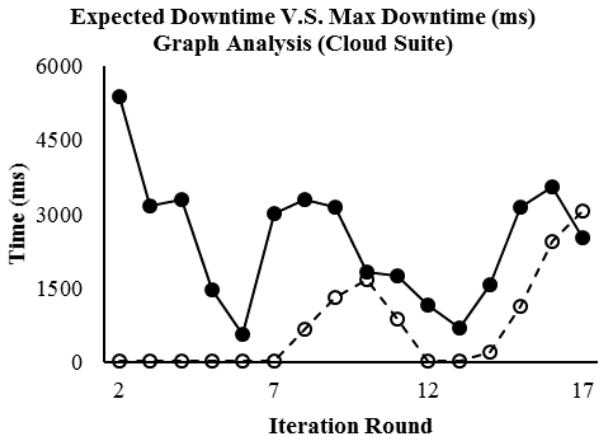

Figure 9. Expected Downtime V.S. Max Downtime in case of Graph Analysis (Cloud Suite) workload, with Adaptive VM Downtime Control (–●–: Expected Downtime – –○– –: Max Downtime)

for enough long time after a previous migration), or the downtime control method would not successfully detect the change of WWS in other special workload sets. For the memory compaction technique, we require to design a method to anticipate whether the migration could gain benefit from this technique. Besides, the proposed downtime control method has many possibilities of optimization. We leave those explanation issues to future work.

ACKNOWLEDGMENT

This research was supported by the IT R&D program of MSIP/KEIT, [K1400011001, Human Friendly Devices (Skin Patch, Multi-modal Surface) and Device Social Framework Technology].

REFERENCES

[1] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, K. Shin, "LiteGreen: Saving Energy in Networked Desktops Using Virtualization," USENIX ATC, 2010.
[2] J. Reich, M. Goraczko, A. Kansal, J. Padhye, "Sleepless in Seattle No Longer," USENIX ATC, 2010.
[3] K. Ibrahim , S. Hofmeyr , C. Iancu , E. Roman, "Optimized pre-copy live migration for memory intensive applications," Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011.
[4] W. Hu, A. Hicks, L. Zhang, E. Dow, V. Soni, H. Jiang, R. Bull, J. Matthews, "A Quantitative Study of Virtual Machine Live Migration," CAC 2013.
[5] C. Jo, B. Egger, "Optimizing Live Migration for Virtual Desktop Clouds," VEE, 2013.
[6] E. Park, B. Egger, and J. Lee, "Fast and space efficient virtual machine checkpointing," VEE 2011.
[7] A. Rai, R. Ramjee, A. Anand, "MiG: Efficient Migration of Desktop VMs using Semantic Compression," USENIX ATC 2013.
[8] A. Kochut and H. Shaikh. "Desktop to cloud transformation planning," IEEE IPDPS, May 2009.
[9] B. Hudzia, P. Svard, J. Tordsson, E. Elmroth, "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines," VEE, 2011