ISSN 2047-3338

Improved Live VM Migration using LRU and Splay Tree Algorithm

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

***Abstract*—Live migration is a technology with which an entire running virtual machine (VM) is moved from one physical machine to another. Migration at the level of an entire VM means that in-memory state can be transferred in a consistent and efficient fashion. In pre-copy approach that is mainly used in live migration, total migration time which affect on the performance of VM, is prolonged by iterative copy operations and the significant amount of transferred data during the whole migration process. In this paper, we presented a framework that includes pre-processing phase in traditional pre-copy based live migration for reducing the amount of transferred data. In pre-processing phase, we propose the prediction working set algorithm. Applying the proposed algorithm which combined LRU (Least Recent Used) cache with splay tree algorithm, the system can reduce the amount transferred memory page. We evaluated the effectiveness of the working set prediction algorithm with various workloads. Experiment demonstrates that compared with XEN's default pre-copy based migration algorithm, the proposed framework can reduce 23.67% of the total data transferred during migration and 11.45% of total migration time on average.**

***Index Terms*—Least Recently Used (LRU), Pre-Copy Based Live Migration, Total Migration Time and Virtual Machine**

I. INTRODUCTION

VIRTUALIZATION technology is gaining more and more interest in the high-performance computing world. It has many advantages, such as security isolation, abstraction from heterogeneous hardware, reliability, and so on. Virtualization technology also 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 most important features of virtualization technology. It allows a clean separation between hardware and software, and facilitates fault management.

Live migration of VMs 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.

There are two major performance metrics of migration: i) total migration time, the duration between the start of migration and the time when migrated VM begins to run in destination and gets a consistent state with the original one; ii) downtime, the time period from when migrated VM is suspended on the source node to when it is resumed on the destination node in the last migration phase.

The key challenge is to achieve impressive performance with minimal service downtime and total migration time in live migration [1]. 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 VM's memory content, storage content, and network connections from the source node to the target node.

The best technique for live migration of VMs is pre-copy [2]. 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 VM, 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. 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.

This paper aims to predict the memory pages which are used in near future. Recency and Frequency of references are the two important parameters that determine the likelihood of a memory page to be accessed in the near future. The LRU policy gives importance only to the recency of references.

Making memory pages grouping is very important to predict the memory pages. It is also need to measures how close a group of data is accessed together within an execution. They measure togetherness with a stack distance, which is defined as the amount of distinct data accessed between two

memory references in an execution trace. We used splay tree algorithm for grouping the current memory pages.

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 VMs and then define working set. We also present the proposed framework for pre-copy based live migration to reduce the total migration time. According to the experimental results, we can reduce the total migration time of live migration of VMs.

The rest of this paper is organized as follows. Section 2 describes the related work. In Section 3, we discuss the live migration of VMs, LRU replacement algorithm and Splay Tree algorithm. In Section 4, the framework of pre-copy based live migration and proposed predicting working set algorithm are described. In Section 5, experimental results are described. Finally, Section 6 concludes the paper.

II. Related Work

In live VM migration, Pre-Copy [2] is the default migration algorithm for Xen. Because of programs' local principles, VM's downtime is expected to be minimal, and in the same time, the source node maintains the newest memory image until migration is finished. The whole process is reliable, because if destination node crashes, Pre-Copy can abort migration and continue to run VM on the source node. However, when applications' loads are intensive, Pre-Copy has to transfer too much memory image data, and consequently has great time overhead.

Post-Copy [3] is proposed to solve this problem, and it works in two phases. In the first phase, VM is suspended on the source node, and its VCPU context and minimal memory working set are copied to destination. In the second phase, VM is started on the destination node, and all the memory write operations are executed locally.

When VM needs to read some pages that the source node has the newest version, these pages are fetched through network. In the meantime, the source node keeps pushing remaining memory image to the destination node until done. Post-copy thus ensures that each memory page is transferred at most once. However, both source and destination have part of the newest memory status during migration. If the destination node crashes, VM can not restart on the source node, so Post-Copy does not have the same level of reliability as Pre-Copy.

In pre-copy based live migration [4] of VM such as VMware [5], XEN [6] and KVM [7] and post-copy based live migration, sorting the page-cache in LRU order performs better than non-LRU cases by improving the locality of reference of neighboring memory pages in the pseudo-paging device. Recency and Frequency of references are the two important parameters that determine the likelihood of a memory page to be accessed in the near future. The LRU policy gives importance only to the recency of references. FBR is a frequency-based policy that is similar to LRU but uses the concepts of correlated references [8]. In LRU-K policy, replacements are based on the time of the K^{th} to last non-correlated reference to each block [9].

The proposed algorithm used the LRU replacement policy with splay tree for grouping the current memory pages according to their process ID to predict the working set in Live VM migration.

III. Background Theory

A. Live Migration of VMs

VM migration takes a running VM 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 VM.

Live Migration migrate OS instances including the applications that they are running to alternative VMs freeing the original VM for maintenance. It rearranges OS instances across VMs in a cluster to relieve load on congested hosts without any interruption in the availability of the VM as shown in Fig. 1.

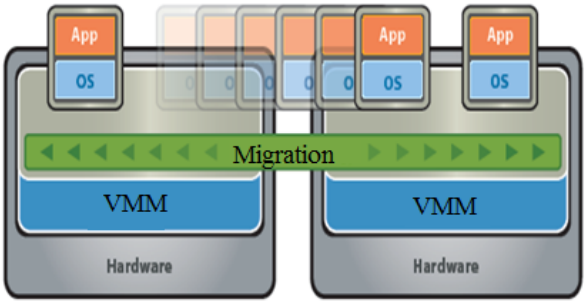


Fig. 1. Live Migration of VMs

A key challenge in managing the live migration of OS instances is how to manage the resources which include networking, storage devices and memory [9].

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 VM migration. Moving the memory instance of the VM from one physical state to another can be approached in any number of ways.

In pre-copy based live migration of VM, 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 [5], XEN [6] and KVM [7] is used the pre-copy approach for live migration of VMs.

B. 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 [10]. The problem is to determine and order for the fames defined by the time of last use. Two implementations are feasible: Counters and Stack [11]. 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 VM.

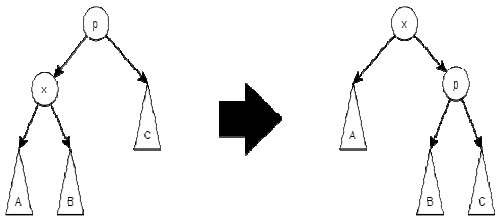
C. Splay Tree Algorithm

A splay tree [12] is a self-adjusting binary search tree with the additional property that recently accessed elements are quick to access again. It performs basic operations such as insertion, look-up and removal in O(log n) amortized time, n means number of nodes. For many sequences of nonrandom operations, splay trees perform better than other search trees, even when the specific pattern of the sequence is unknown.

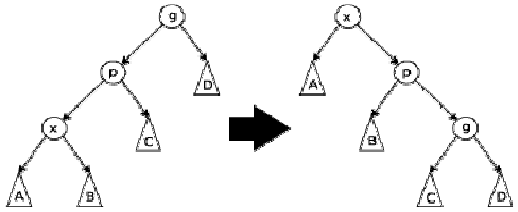
They use a heuristic restructuring called splaying to move a specified node to the root of the tree via a sequence of rotations along the path from that node to the root. Thus, future calls to this node will be accessed faster. Splay trees even enjoy a constant operation time. The key to a splay tree is, of course, the restructuring splay heuristic [12]. Specifically, when a node is searched, it repeats the following splay step until is the root of the tree.

- Case 1 (zig): if p(x), the parent of x, is the root, rotate the edge joining x with p(x).
- Case 2 (zig-zig): if p(x) is not the root, and x and p(x) are both left or both right children, rotate the edge joining p(x) with its grandparent g(x) and then rotate the edge joining x with p(x).
- Case 3 (zig-zag): if p(x) is not the root and x is a left child and p(x) a right child or vice versa, rotates the edge joining x with the new p(x).

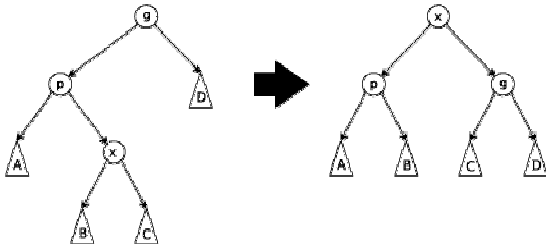
Fig. 2 demonstrates the splay step through examples. Observe the move-to-root heuristic, which moves the target item to the root of the tree for each search operation. Although the cost is high for an individual operation, a benefit is the rough halving of the depth along the access path.



(a) zig rotation case



(b) zig-zig rotation case



(c) zig-zag rotation case

Fig. 2. Splaying steps, the node accessed is x.

Good performance for a splay tree depends on the fact that it is self-optimizing, in that frequently accessed nodes will move nearer to the root where they can be accessed more quickly. The worst-case height - though unlikely - is O(n), with the average being O(log n). Having frequently-used nodes near the root is an advantage for nearly all practical applications.

The advantages of splay tree algorithm are i) Simple implementation, ii) Comparable performance, iii) Small memory footprint, and iv) Working well with nodes containing identical keys. The disadvantage of splay tree algorithm is the height of a splay tree can be linear.

IV. A Framework for Live VM Migration

In this Section, we present a 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 Fig. 2.

i) 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 splay tree algorithm to define the working set list that collects the most recent used memory pages (operation 1).

ii) 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 (operation 2).

iii) Stop and Copy Phase

This phase consists of three steps. Firstly, the systems suspend the source VM for a final transfer round (operation 3). Secondly, the system discards the source VM and then transfer last modified pages and CPU state

(operation 4). Lastly the system activates the Target VM (operation 5).

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.

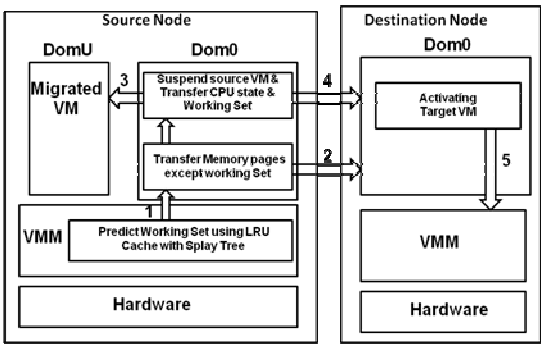


Fig. 3: A Framework for pre-copy based live migration

A. Proposed 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. The proposed algorithm predicts the most recently used memory pages that directly affects the total migration time.

In the proposed algorithm as shown in Figure 4, memory of VM employ with LRU cache with splay tree. The page which is at the top of the LRU cache and its splay tree which collect memory pages used in the same process are defined as Working Set.

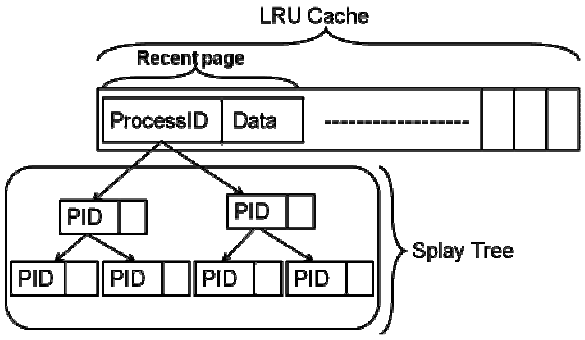


Fig. 4. Block diagram for LRU cache with Splay tree

If a new page from the new process is faulted into main memory, it is constructed the splay tree with the new ProcessID. If a new page from the running process is faulted, it is inserted into the splay tree with the correspondence ProcessID. After that, the new page with its correspondence splay tree is placed at the top of the LRU cache and defined as the Working Set. If the LRU cache is full, the last page is removed from the LRU cache to place the most recent used page. The Working Set prediction is deployed by least recently used (LRU) algorithm and splay tree algorithm as shown in Fig. 5. The proposed algorithm:

- predicts the working set 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.
- dynamically defines the working set according to the most recent process.
- reduces the transferring of the zero page's reference page by grouping the memory pages with their ProcessID.

Prediction time is performance bottleneck of addition overhead introduced by prediction operation. We use Least Recent Used (LRU) cache with splay tree prediction algorithm to get the perfect accuracy of prediction operation without a notable execution time.

```
Algorithm: Working Set Prediction
(1)  Input : Request page j with ProcessID
(2)  {
(3)   SplayTree(ProcessID, Requestpage );
(4)   if (! LRU cache is full)
(5)   {
(6)    place the page j  with splay tree at the top of the LRU   cache;
(7)   }
(8)  else
(9)   {
(10)   if ( !request page j is in LRU cache)
(11)  {
(12)       remove page i whose is located  at the end of the LRU
           cache;
(13)       insert request page j with splay tree at the top ofthe LRU
           cache;
(14)     }
(15)  else
(16)  {
(17)     move the page j  with splay tree at the top of the LRU
         cache;
(18)  }
(19) }
(20)  define the page j  with splay tree as the Working Set;
(21) }

   Function: SplayTree (ProcessID, Requestpage)
(1)  {
(2)  if ( ! ProcessID )
(3)   {
(4)    construct  new Splay tree with page j;
(5)  }
(6)   else( ! page j in correspondent Splay tree)
(7)   {
(8)    insert  page j in correspondent splay tree;
(9)   }
(10) }
```

Fig. 5. Proposed Working Set Prediction Algorithm

V. EVALUATION

In this section, we evaluate the framework with the proposed working set prediction algorithm by using various workloads, then present and analyze performance improvement compared with XEN's default Pre-Copy based migration algorithm.

A. Experimental Environment

Our experiment platform is a cluster composed by six identical computer servers. One server works as the storage server, and provides shared storage by iSCSI protocol through isolated gigabit Ethernet to other two servers, which act as the source and destination of migration separately. The remaining three servers work as clients for different workloads. For each server, its configuration includes two Intel Xeon E5520 quad-core CPUs running at 2.2GHz, 8GB DDR RAM. All the servers are connected by a Gigabit LAN. The version of VMM is Citrix XEN-5.6.0 and Guest OS is the modified Linux-2.6.18.8. The migrated unprivileged VM is configured with one VCPU and 512MB RAM. Migration does not use separate network, it shares the same network with workloads.

For evaluating the performance of our proposed framework, we select several representative applications in virtualization environment as the workloads of migrated VM:

- *Compilation:* Linux kernel compilation with two parallel threads, which is a balanced workload to test performance of system virtualization.
- *VOD:* Server for Video on Demand, which contains little writes, but it is sensitive to latency. There are 300 concurrent sessions connected from clients.
- *Dynamic Web Server:* Specweb2005 which uses the most objective workloads (Banking and Ecommerce) for measuring a system's ability to act as a synthetic web server. It supports static and dynamic web contents. It consumes too much system resources, and is a more challenging workload for migration. It works in HTTP 1.1 protocol. There are 300 concurrent sessions connected from clients.

Migrated VM is the only unprivileged VM running in source, and there are not any other unprivileged VMs residing in destination.

B. Transferred Memory Page

We illustrated the number of transferred memory pages of each migration round in Complication workload during migration process in Fig. 6.

In Fig. 7, we present the number of transferred memory pages of each migration round in Video on Demand (VOD) workload during migration process.

In Fig. 8, the graph shows the number of transferred memory pages of each migration round in Banking workload during migration process.

In Fig. 9, we present the number of transferred memory pages of each migration round in Ecommerce workload during migration process.

According to the results, the number of transferred memory pages using the LRU algorithm as working set prediction is not significantly different with XEN except at the first round of migration process but the propose system (LRU cache with splay tree algorithm using as working set prediction) reduce the transferred memory pages in every round of migration process.

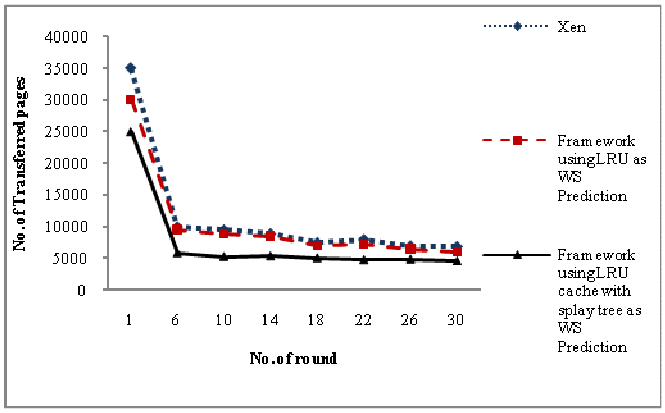


Fig. 6. Number of Transferred pages of each migration round whose workload is Complication

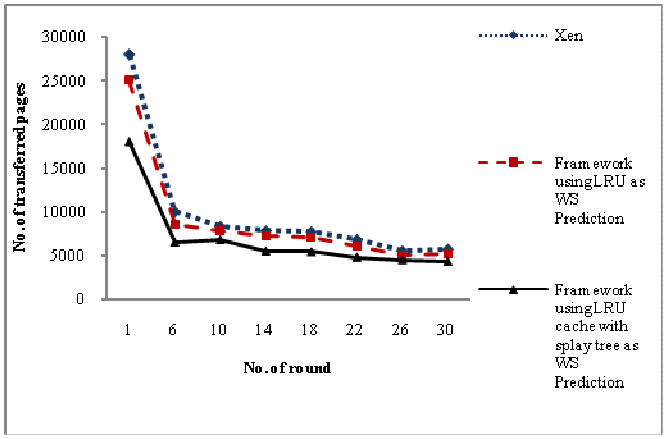


Fig. 7. Number of Transferred pages of each migration round whose workload is Video on Demand

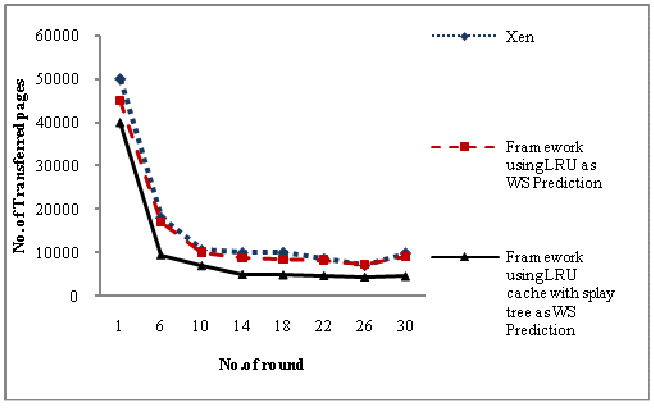


Fig. 8. Number of Transferred pages of each migration round whose workload is Banking

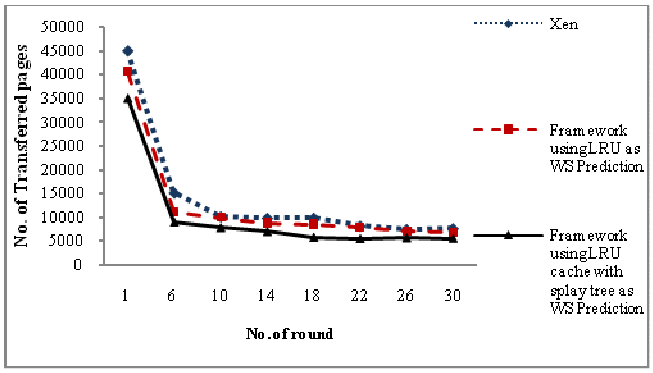


Fig. 9. Number of Transferred pages of each migration round whose workload is Ecommerce

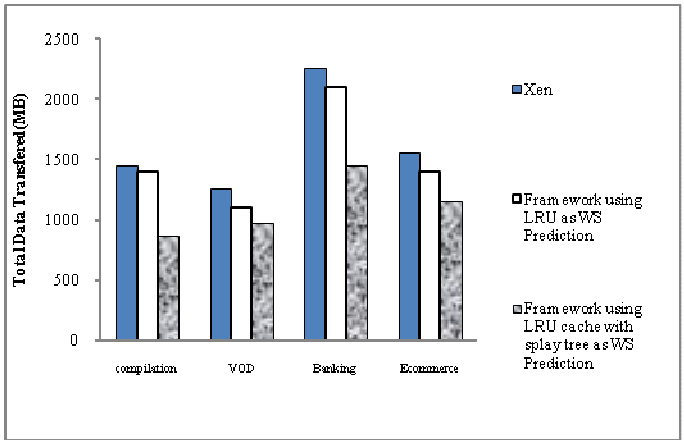


Fig. 10. Total Data Transferred in Migration Process

In Fig. 10, the result shows the total transferred memory size(MB) for the proposed framework (using LRU cache with splay tree as working set prediction), the framework (using LRU as working set prediction) and XEN with various workloads. These results show that the workload which runs the same process repeatedly such as Banking can more reduce the total transferred memory size using the proposed working set algorithm during migration process.

C. Total Migration Time

We analyze the live migration process and calculate the total migration time of the propose framework and XEN. Xen has a built-in migration tool which uses pre-copy scheme for runtime state migration. It requires a shared device to provide file system for the migrated virtual machine.

Assuming that the memory could be processed in the propose framework as pages with a fixed size equal to *P* bytes, *M* is the memory size, the bandwidth available for the transfer is constant and equal to *B* bytes per second, W_S is the working set list and W_M is the modified page per second as shown in Table 1.

Therefore, the time cost to transfer the memory pages in the propose framework is shown in the following.

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

TABLE I. 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*

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

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= $T_{pre} + \sum_i^n T_i + T_{stop}$ (4)

And the time cost to transfer the memory pages in XEN is

Execution time for *push phase* of XEN :

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

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 XEN,

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

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

In push phase of the proposed framework as shown in eq. (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 eq. (5) for the push phase of the XEN, 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.

The result shows in Fig. 11 is the comparison of total migration time for the proposed framework (using LRU cache with splay tree as working set prediction), the framework (using LRU as working set prediction) and XEN with various workloads. According to these results, we can reduce 11.45% of total migration time on average in live migration of virtual machine than XEN.

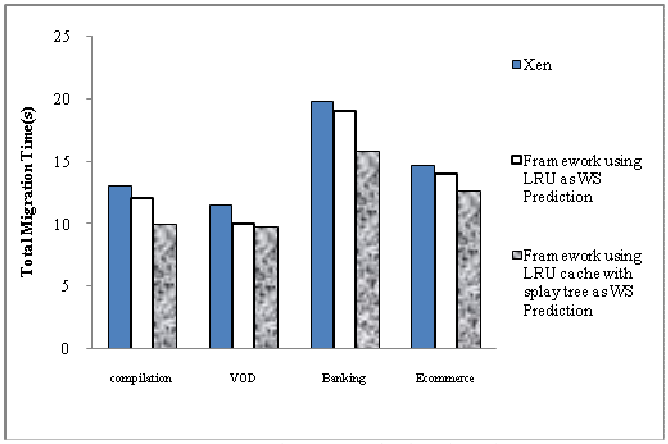


Fig. 11. Total Migration Time

VI. Conclusion

Pre-copy migration ensures that keep downtime small by minimizing the amount of VM state. It provides to abort the migration should the target node ever crash during migration because the VM is still running at the source. It also needs to reduce the total migration time and impact of migration on performance of source and target VMs. We then presented the design and implementation of the proposed framework, which introduces working set prediction during migration, and reduces total migration time 11.45% on average compared with Xen's default migration algorithm.

In future, we will study the compression algorithm of memory data and further reduce the total transferred memory pages in the migration process to make our proposed system more attractive.

References

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

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

[3] M. R. Hines and K. Gopalan, "Post-copy based live VM 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.

[4] P. Lu and K. Shen, "VM 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.

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

[6] 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.

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

[8] J. T. Robinson and N. V. Devarakonda. Data Cache Management Using Frequency based Replacement. In the Proceedings of the 1990 ACM SIGMETRICS Conference, pages 134-142, 1990. [11] A.Subramanian, "An Explanation of Splaying", Academic Press. Inc, 1996.

[9] E. J.O'Neil, P. E. O'Neil, G. Weikum. The LRU-K Page Replacement Algorithm For Database Disk Buffering". In Proceedings of the 1993 ACM SIGMOD Conference pp. 297-306, 1993.

[10] K. Cheng and Y. Kambayashi. LRU-SP: A Size-Adjusted and Popularity-Aware LRU Replacement Algorithm for Web Caching.

[11] 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.

[12] D. Dominic Sleator, R. Endre Tarjan. Self-Adjusting Binary Search Trees. In Journal of the Association for Computing Machinery, Vol. 32, No.3, July 1985, pp.652-686.