Working Set Prediction using LRU cache with Splay Tree Algorithm in Live VM Migration

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

Abstract

Live virtual machine (VM) migration provides great benefits for load balancing, power management, fault tolerance and other system maintenance issues in modern data centers. VM live migration basically consists of transferring its memory pages from a source server to a destination server. The amount of transferred memory pages affected the downtime and application performance of virtual machines. In pre-copy approach in live VM migration, total migration time is prolonged the significant amount of transferred data during the whole migration process. In this paper, we propose a working set prediction algorithm which combine LRU (Least Recently Used) cache with splay tree algorithm. Deploying splay tree algorithm with LRU, the proposed algorithm predict the memory pages of the most recent process and define as the working set. Applying the proposed algorithm, we can reduce the amount of transferred memory pages during migration process by transferring the working set in last round. Experiment demonstrates that compared with XEN's default pre-copy based migration algorithm, the proposed algorithm can reduce 23.67% of the total transferred memory pages during migration process.

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

1. 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.

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 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 [9]. 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.

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 [13]. 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 frequently updated memory pages, in pre-copy based VM migration and then define as the working set. According to the experimental results, the proposed algorithm can reduce the total transferred memory pages in live migration of virtual machines by transferring the working set in last round of the migration process.

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, LRU and Splay Tree algorithm. In Section 4, the proposed algorithm of pre-copy based live migration and in Section 5, experimental results are described. Finally, Section 6 concludes the paper.

2. Related Work

In live VM migration, Pre-Copy 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 [13] 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. 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. FBR is a frequency-based policy that is similar to LRU but uses the concepts of correlated references [10]. In LRU-K policy, replacements are based on the time of the Kth to last non-correlated reference to each block [8].

In 2Q Replacement policy, there is a special buffer called the A1 queue into which a missed block is initially placed. A block in the A1 queue is promoted to the main cache only when it is re-

referenced while in the A1 queue. Otherwise it will be evicted when it becomes the LRU block in the A1 queue [19]. LRFU (Least Recently / Frequently Used) replacement policy subsumes LRU and LFU replacement strategies [1]. This policy takes the advantage of LFU and LRU replacement schemes. LRU-SP is a size adjusted and popularity-aware extension to Least Recently Used (LRU) for caching web objects [11]. In this paper we propose working set prediction algorithm which deploy the LRU replacement policy and Splay Tree algorithm.

Current computers usually use cache blocks, making data grouping very important. Before grouping data, we must first define a relation Zhong, et al [5, 20] defined a relation called reference affinity, which 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.

The proposed algorithm used splay tree for grouping the current memory pages according to their proceessID.

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.

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. The proposed algorithm applies on the memory migration process which affects the performance of virtual machines.

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 [12], XEN[3] and KVM [8] is used the pre-copy approach for live migration of virtual machines. OpenVZ [9] also use this approach for OS-level migration.

3.2. 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.

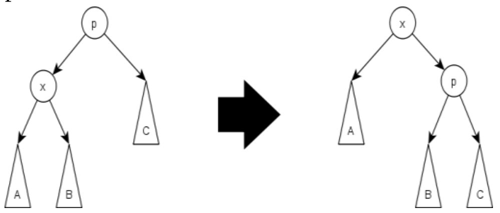
3.3. Splay Tree Algorithm

A splay tree 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.

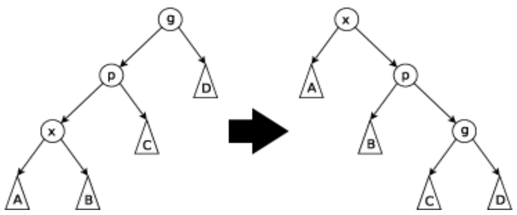
They use a heuristic restructuring called splaying to move a specified node to the root of the tree via a sequence of rotations along the path from that node to the root. Thus, future calls to this node will be accessed faster. Splay trees even enjoy a constant operation time The key to a splay tree is, of course, the restructuring splay heuristic. 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).
- Case3 (zig-zag) : if p(x) is not the root and x is a left child and p(x) a right child or vice versa, rotate the edge joining x with the new p(x).

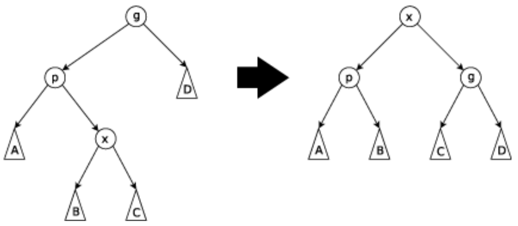
Figure 1 demonstrates the splay step through examples. Observe the move-to-root heuristic, which moves the target item to the root of the tree for each search operation. Although the cost is high for an individual operation, a benefit is the rough halving of the depth along the access path.



(a) zig rotation case



(b) zig-zig rotation case



(c) zig-zag rotation case

Figure 1: 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.

4. Proposed System Design

In this Section, we present the proposed working set prediction algorithm for pre-copy based live migration to reduce the total transferred memory pages. The proposed algorithm run on virtual machine monitor (VMM) to predict the working set as shown in Figure 2.

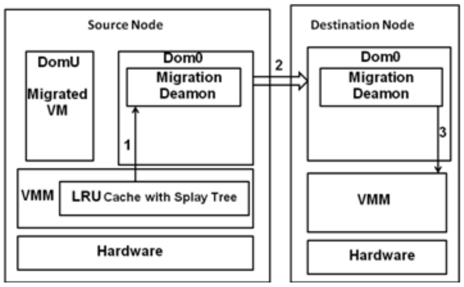


Figure 2: Proposed System Design

The system run migration deamon in privileged domain (Dom0) of the source of migration maps migrated VM's memory pages according to the LRU cache using Splay Tree Algorithm(operation 1). 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.

Then 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).

At last, the system suspends the source virtual machine for a final transfer round. It also transfer last modified pages and CPU state and then discards the source VM and activates the Target VM(operation 3).

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 transferred memory pages. 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.

4.1. Proposed LRU cache with Splay Tree 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 which predicts the most recently used memory pages that directly affects the total migration time. 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 using ProcessID is deployed by least recently used (LRU) Algorithm and splay tree algorithm as shown in Figure 3.

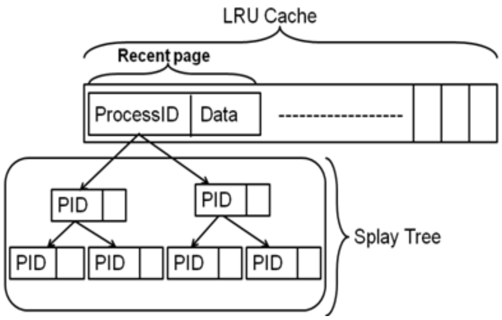


Figure 3: Block diagram for LRU cache with Splay tree

```
Algorithm: Working Set Prediction
Input : Request page j with ProcessID
Begin
1.  SplayTree(ProcessID, Requestpage );
2.  if the LRU cache is full then
3.      place the page j with splay tree at the top of the LRU cache;
4.  else
5.      if the request page j is not in LRU cache then
6.          remove page i whose is located at the end of the LRU cache;
7.          insert request page j with splay tree at the top of the LRU cache;
8.      else
9.          move the page j with splay tree at the top of the LRU cache;
10.     endif
11. define splay tree with the page j as the Working Set;
12. endif
End

Procedure: SplayTree (ProcessID, Requestpage j)
Let PID: ProcessID;
    j: Requestpage;
    p: Parent of Requestpage j;
    g: Grandparent of Requestpage j;
Begin
1.  if PID is not already exist in memory then
2.      construct j as the root node of new Splay Tree with PID;
3.  else
4.      if j is the root node of Splay Tree with PID then
            return Splay Tree with PID;
5.      elseif the p is the root node
6.          if the j is the left child of p then
                perform rightrotation of p;
7.          else the j is the right child of p then
                perform leftrotation of p;
8.          endif
9.      elseif p is the left child of g
10.         if j is the left child of p then
                preform rightrotation of g and p;
11.         else j is the right child of p then
                preform leftrotation of p and rightrotation of g ;
12.         endif
13.     elseif p is the right child of g
14.         if j is the right child of p then
                preform leftrotation of g and p;
15.         else j is the left child of p then
                preform rightrotation of p and leftrotation of g ;
16.         endif
17.     endif
18. endif
19. return j as the root node of Splay Tree with PID
End
```

Figure 4: Proposed Working Set Prediction Algorithm

In the proposed algorithm as shown in Figure 4, memory of virtual machine 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.

5. Evaluation

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

5.1. 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 [21] 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.

In Figure 5, we present the number of transferred memory pages of each migration round in Complication workload during migration process. The processes of this workload may be repeatedly and so the proposed algorithm can predict more accurately than other workloads.

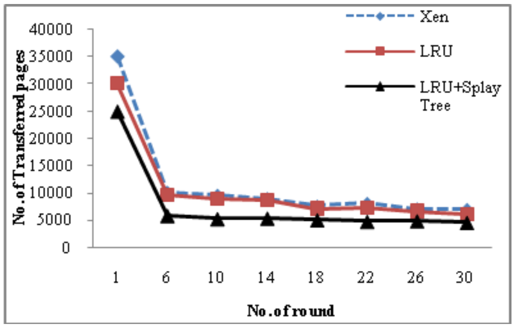


Figure 5: Number of Transferred pages of each migration round whose workload is Complication

The number of transferred memory pages of each migration round in Video on Demand (VOD) workload during migration process is illustrated in Figure 6. The nature of this workload is sequentially accessed the processes and so the prediction result of the proposed algorithm is not much different with using only LRU algorithm.

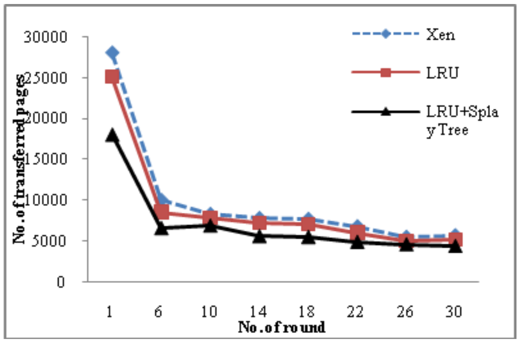


Figure 6: Number of Transferred pages of each migration round whose workload is Video On Demand

In Figure 7, the graph shows the number of transferred memory pages of each migration round in Banking workload during migration process. Because of using the same process repeatedly, the accuracy of the proposed system is higher than other workloads.

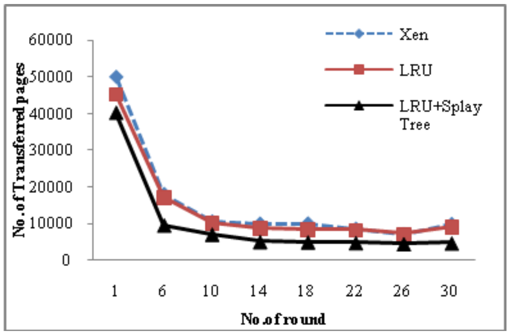


Figure 7: Number of Transferred pages of each migration round whose workload is Banking

In Figure 8, we present the number of transferred memory pages of each migration round in Ecommerce workload during migration process. It not much different with the result of Banking workload.

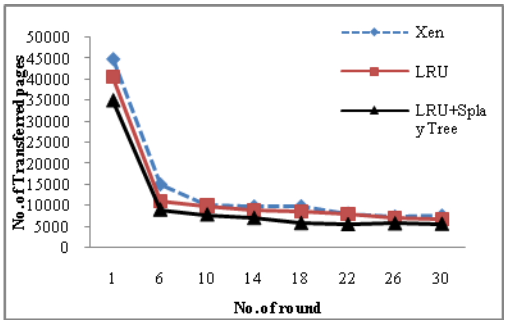


Figure 8: Number of Transferred pages of each migration round whose workload is Ecommerce

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.

In Figure 9, 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.

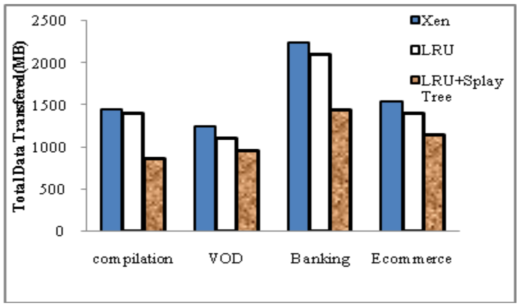


Figure 9: Total Data Transferred in Migration Process

6. 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 present the design and implementation of the proposed framework, which introduces working set prediction during migration. According to the results, we can reduce 23.67% of total data transferred memory pages on average in live migration of virtual machine than XEN.

7. References

[1] A.Subramanian, "An Explanation of Splaying", Academic Press. Inc, 1996

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

[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] 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.

[5] C. Ding and Y. Zhong. Predicting Whole-Program Locality through Reuse-Distance Analysis. *In Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, San Diego, California, June 2003.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] R. Goldberg, “Survey of virtual machine research,” IEEE Computer, pp. 34–45, Jun. 1974.

[17] 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.

[18] 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.

[19] T. Johnson and D. Shasha. 2Q: A low Overhead High Performance Buffer Management Replacement Algorithm. *In the proceedings of the 20th Intl Conference on Very Large Data Bases,* pages 439-450, 1994.

[20] Y. Zhong, M. Orlovich, X. Shen, and C. Ding. Array Regrouping and Structure Splitting Using Whole-Program Reference Affinity. To be appear on ACM SIGPLAN *Conference on Programming Language Design and Implementation*, Washinton DC, June 2004.

[21] http://www.spec.org/web2005/