RPFF: A Remote Page-fault Filter for Post-copy Live Migration

Kui Su, Wenzhi Chen, Guoxi Li, Zonghui Wang
College of Computer Science
Zhejiang University, Hangzhou, 310027, China
Email: {sukuias12, chenwz, guoxili, zjuzhwang}@zju.edu.cn

***Abstract*—Live migration of virtual machine has attracted significant attention in recent years. It facilitates system online maintenance, load balancing, fault tolerance and power management. Existing pre-copy live migration approach has to iteratively copy redundant memory pages, which causes high network overhead and slow migration. Another post-copy live migration approach can provide quick migration with low network overhead but would lead to a lot of remote page faults which greatly degrade application performance. In this paper, we improve the post-copy approach by eliminating unnecessary remote page faults. In our improved post-copy approach, page faults caused by guest's overwriting operations on remote memory pages are intercepted and handled by allocating new local memory pages, instead of fetching memory pages from the source host across the network. We have implemented our approach into the KVM/QEMU hypervisor and ran a series of experiments with Linux guests. The experimental results demonstrate that our improved approach performs much better than traditional post-copy approach.**

***Keywords*-Virtualization; Live Migration; Page-overwrite; Post-copy; Remote Page-fault;**

I. Introduction

In recent years, Live migration has become a key selling point for state-of-the-art virtualization [1, 2] technologies. It refers to techniques in which a VM is moved from one host to another with almost zero downtime and without interrupting services running in VM. Live migration is a powerful tool for online system maintenance, fault tolerance, workload balancing, testing and consolidation of VMs, etc.

Existing migration schemes can be broadly classified as either pre-copy schemes [3, 4] or post-copy schemes [5], based on when the VM context and its memory image are transferred. In pre-copy live migration, all states of a VM are completely copied to a destination host before the execution host is switched to the destination. Updated memory pages during memory copy are iteratively copied to the destination. It takes a long time to switch the execution host of an actively running VM, and it is hard to estimate when migration is completed. If a transmitted page is subsequently dirtied, it is resent in the next round. When encountered with write-intensive workloads, this strategy will cause a lot of repeated transmissions, which waste network bandwidth resources and can not improve the downtime.

On the contrary, the post-copy migration schemes transfer the VM context to the destination node immediately, resume the VM execution with the memory state still present at the source node, and then initiate a background memory copy process so that the VM context at the destination node finds requested memory pages in its local memory. If the VM context does not find a requested memory page in its local memory, it triggers a remote page fault handler to obtain the page from the source node. Therefore, if a workload is not memory intensive or the background copy delivers requested pages in time, the post-copy scheme does not expose the machine downtime to the user. The main problem of the post-copy scheme is that many remote page faults incur a significant performance loss. If a workload is memory intensive and its memory access patterns do not exhibit spatial and temporal localities, the background memory copy process becomes ineffective even though it consumes all the network bandwidth available.

In this work, the goal is to improve the performance of write-intensive workloads in post-copy live migration by reducing the number of remote page faults. According to our observation, memory management performed by OS includes activities like zeroing pages before they are (re)allocated [6], copying memory pages on write (COW) [7], and migrating pages from one DRAM location to another due to memory compaction [8], e.g., for super paging [9, 10]. Whether by copying memory or zeroing it, OS often overwrite full pages without regard to their old content. Such activity has no undesirable side effects in local machines. But during post-copy migration, the target page being overwritten might be required and triggering a remote page fault. Therefore, we have designed a filter to eliminate such remote page faults. In our work, the filter intercept and emulate the guests write instruction to identify overwriting operations on remote memory pages, then the filter redirects such operations to new allocated local pages so that the remote page fault is eliminated.

The contribution of this paper is that this is the first work to reduce remote page faults caused by guest's page-overwrite operations in post-copy migration. We have implemented our approach into the KVM/QEMU hypervisor [11] and ran a series of experiments with Linux guests. The experimental results demonstrate that our improved approach performs much better than traditional post-copy approach.

The remainder of this paper is organized as follows: Section II gives an overview of related work on live migration of virtual machines. Section III contains the design and

implementation of the remote page-fault filter. Section IV evaluates our approach with a variety of workloads. Finally, we summarize our contributions and outline our future work in section V.

II. Related Work

Many techniques for live virtual machine migration were introduced in recent years. Pre-copy is the most common and predominant approach for live VM migration. It is employed by many popular hypervisors such as VMware [12], KVM , Xen, and VirtualBox [13]. which 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.To reduce the downtime of the VM, the state of the VM is copied in several iterations [3]. While transferring the state of the last iteration, the VM continues to run on the source machine. When applications writable working set becomes small, the virtual machine is suspended and only CPU state and dirty pages in the last round are sent out to the destination. The pre-copy approach achieves a very short downtime in the best case, but for memory-write-intensive workloads the stop-and-copy phase may increase to several seconds. Remote Direct Memory Access on top of modern high-speed interconnects can significantly reduce memory replication during migration [14].

Another novel strategy post-copy is also introduced into live migration of virtual machine [5]. It takes the opposite approach: first, the VM is stopped on the source host and the state of the VCPU and devices is transferred to the target host. The VM is immediately restarted on the target host. Memory pages are fetched on-demand from the source machine as the VM incurs page-faults when accessing them on the target machine. This approach consumes a shorter downtime but produces a longer total migration time, and the performance during the migration is likely to be considerably degraded when a large number of memory pages have been demand-paged across the network. Hines et al. [5] combine post-copying with dynamic self-ballooning and adaptive pre-paging to reduce both the amount of memory transferred and the number of page faults. Hirofuchi et al. [15] employ a post-copy-based approach to quickly relocate VMs when the load of a physical host becomes too high.

Other techniques include live migration based on trace and replay [16], memory compression [17, 18], simultaneous migration of several VMs from one host to another [19], or partial VM migration [20]. Liu et al. [16] propose a live migration algorithm called CR/TR-Motion that is based on checkpointing/recovery and trace/replay technology. Their algorithm sends the logs of execution trace instead of memory pages to achieve good migration efficiency for both LAN and WAN environments. Jin et al. [17] propose using adaptive compression of migrated data: different compression algorithms are chosen depending on characteristics of memory pages. They first use memory compression to provide fast VM migration, and they also design a zero-aware characteristics-based compression (CBC) algorithm for live migration. Svard et al. [18] extend this idea by delta-compressing changes in memory pages. Jo et al. [21] propose a technique to reduce the migration time while keeping the downtime to a minimum. They track the I/O operations between the VM and the NAS to maintain a map of the pages that reside on the storage device. For these pages, the memory-to-disk map is transmitted instead of the data itself. So these pages can be directly obtained from the NAS after the map is transmitted.

The work most closely related to our technique was presented by Nadav et al. [22]. They propose the false reads preventer to improve the I/O performance of the hosts when they fall back on uncooperative swapping and/or operate under changing load conditions. We use the same technique as Nadav to transparently intercept and emulate memory write requests from guest OS to identify page-overwriting operations in post-copy live migration and obviously reduce the number of remote page faults.

III. Design and Implementation

In this work ,we propose RPFF, a remote page-fault filter for post-copy based live VM migration. The filter eliminates unnecessary remote page faults by redirecting guest's page requests of overwriting operations to new allocated local memory. In post-copy live migration with RPFF, the number of remote page faults is definitely reduced and the overall performance for memory write-intensive workloads during migration are greatly improved. This section introduces the motivation and describes the design and implementation of RPFF .

A. Motivation

Memory page overwriting events routinely happen in modern operating systems which cache data from permanent storage in unused volatile memory to hide the long access latency. The longer the system is running, the bigger an amount of otherwise unused memory is dedicated to this cache. At this time, once a new process launched, the OS is likely to allocate pages to the new process from the cache. And the allocated pages may be overwritten by the new process regardless of the old content. Moreover, while running write-intensive workloads such as kernel compiling and data compression, a number of overwriting operations on memory pages are executed to store temporary generated data. We conduct an experiment to run the popular write-intensive applications in virtual machines, and achieved the average number of page-overwrite operations in every minute. Figure 1 shows the results of our experiments with Linux guests running four write-intensive workloads on a VM with 2 GB of RAM. Page-overwrite is very common in memory write-intensive applications. Furthermore, memory

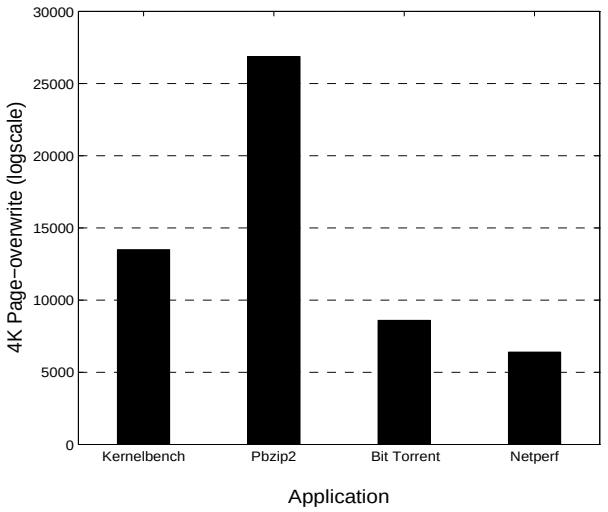

Figure 1: The average number of page-overwrite operations for applications

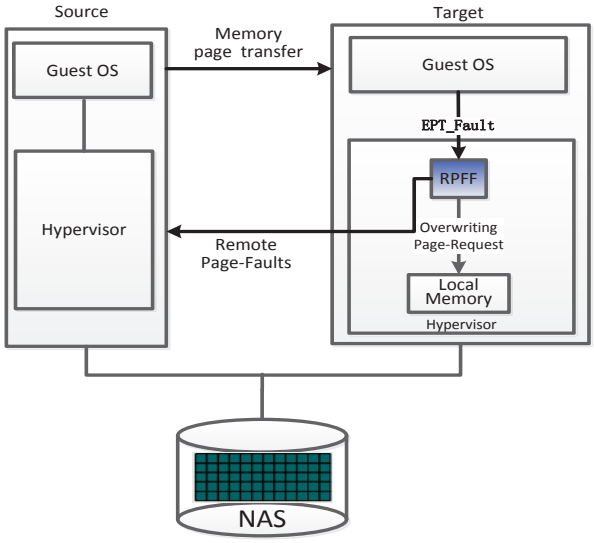

Figure 2: The architecture of post-copy live migration with RPFF

management performed by OS includes activities like zeroing pages before they are (re)allocated [6], copying memory pages on write (COW) [7], and migrating pages from one DRAM location to another due to memory compaction [8], e.g., for super paging [9, 10]. Whether by copying memory or zeroing it, OS often overwrite full pages without regard to their old content.

During post-copy live migration, When the migrated context does not find the requested memory page for overwriting activities in its local memory due to either inaccurate or slow page deliveries, it triggers a remote page fault to bring up the page from the source node through the network. But it is no sense to handle such remote page faults due to that those memory pages would be immediately overwritten on the target without regard to their old content. As is known to all, a number of remote page faults may consume much network bandwidth and significantly degrade the performance of live migration because of the network latency. To address this problem, we propose a filter to eliminate such unnecessary remote page faults to improve the overall performance of post-copy live migration.

B. Architecture

Figure 2 shows the overview architecture of post-copy live migration with RPFF implemented on top of KVM hypervisor. In post-copy live migration, the first step is to stop the VM at the source host. Then processor states are sent to destination VM, and the content of virtual CPU registers and the states of devices are copied to the destination. Next to resume the VM at the destination without any memory content. If the VM tries to access pages that have not yet been transferred, the VM is temporarily stopped and the fault pages are demand paged over the network from source. Then The VM is resumed. With RPFF, if the guest OS requests to overwrite a memory page which happens to reside in remote source node, the filter will intercept the request and redirect it to its local memory. Then the new allocated local memory page is returned to the guest OS for overwriting operation. Therefore, the network latency due to the remote page faults are significantly reduced and the performance of the guest's applications is improved.

RPFF does not utilize any knowledge about guest OS internals, nor does it resort to para-virtual guest/host collaboration that others deem necessary [23]. Instead, it optimistically intercepts and emulates guest write instructions directed missing pages, hoping that all bytes comprising the page will soon be overwritten, obviating the need to read the old content from the remote source node. When that happens, the filter stops emulating and repurposes its write buffer to be the guests page.

C. Implementation

We have implemented the proposed technique RPFF in a existing open-source project [24] which disclose post-copy enhancement of QEMU/KVM using a character device and page faulting transport via user mode. RPFF is implemented in QEMU/KVM via kernel mode to intercept and emulate overwriting operations generated by guest's applications.

In RPFF, when the guest access memory pages which are not in the local memory, it triggers a EPT_Fault, in handling the fault, RPFF first analysises the instruction to distinguish write and read instructions. For write instruction, RPFF would emulate them and storing their result in page-sized, page-aligned buffers. If a buffer fills up, RPFF maps it

to the guest, thereby eliminating the extraneous remote page fault. If not, RPFF still trigger a remote page fault to get the required page. To improve performance, the guest is allowed to continue to execute so long as it does not read unavailable data; if it does, then RPFF suspends it. When the required page content finally arrives, RPFF merges the buffered and read information, and it resumes regular execution.

Guest on the target may trigger two type of EPT_Faults. One is resulted from memory pages reside in source node. Such faults should be handled as the above method. But another fault which is caused by local memory request should be handled as the same as in the local since such faults do not trigger remote page faults. Therefore, during the migration, the source node also transfer the EPT content of the guest to the target so that RPFF can identify that if the EPT_Fault is caused by remote memory or local memory pages.

IV. Experimental Evaluation

We have implemented the proposed technique RPFF in a existing post-copy project based on QEMU/KVM. To demonstrate the effectiveness of RPFF, we measure and compare the performance of post-copy migration with and without RPFF in terms of following metrics: (1) Total migration time; (2) Downtime; (3) The number of remote page faults; (4) Performance of running applications.

A. Experimental Setup

We run our experiments on a pair of PCs with Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz with 8GB RAM, an RTL8168 Ethernet NIC, and a single 1TB 7200 RPM SATA disk.The machines were connected through 100Mbps LAN using separate network adapters for the host and the VM networks. The VM images were stored on a NFS share on the source machine. Hosts and guests run Ubuntu 12.04, Linux 3.7. In all the evaluations, there was only one VM running on the source machine and there were no VMs running on the destination machine. We use the following applications to measure the performance of RPFF:

Kernbench: a standard benchmark measuring the time it takes to build the Linux Kernel.

Pbzip2: a parallel implementation of the bzip2 block-sorting file compressor.

Netperf: a benchmark that can be used to measure various aspects of networking performance.

BitTorrent: a simple representative of a multi-peer distributed application.

B. Remote Page Faults

RPFF is proposed to reduce remote page faults which greatly degrade the performance of post-copy live migration. Our first experiment is to measure and compare the remote page-faults in post-copy live migration with and without RPFF. As shown in Figure 3, RPFF reduce the remote page faults in post-copy live migration for all the tested applications. It performs the best while running Pbzip2 application, the remote page faults is reduced by 31%. The others are 15.8%, 10% and 13.2%, respectively. The results demonstrate that RPFF performs well for memory-intensive applications in terms of reducing remote page faults in post-copy live migration.

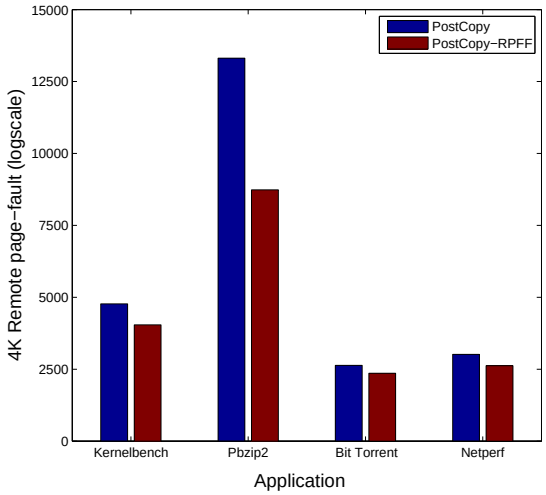

Figure 3: Number of remote page-fault during live Migration

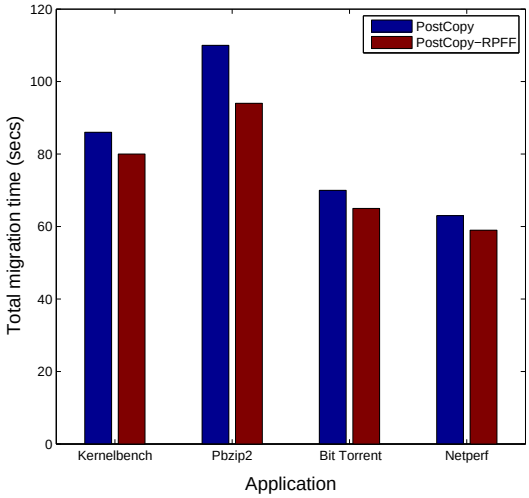

Figure 4: Total migration time for applications

C. Total Migration Time

Figure 4 shows the total migration time for the four application scenarios described in the above. RPFF reduces the total migration time for all the applications. This is not surprising, in post-copy live migration, remote page faults will suspend the application and cause remote page requests

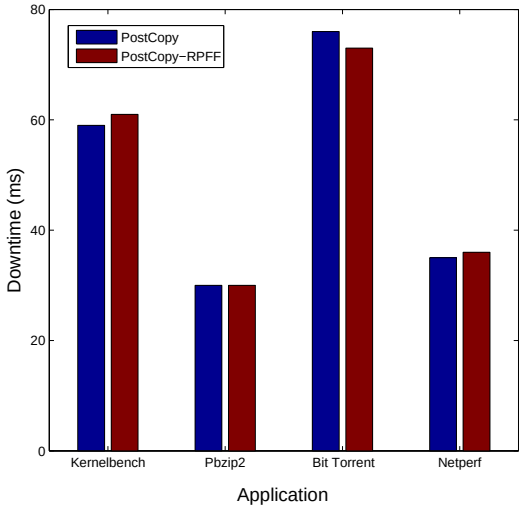

Figure 5: Downtime for different applications during live migration

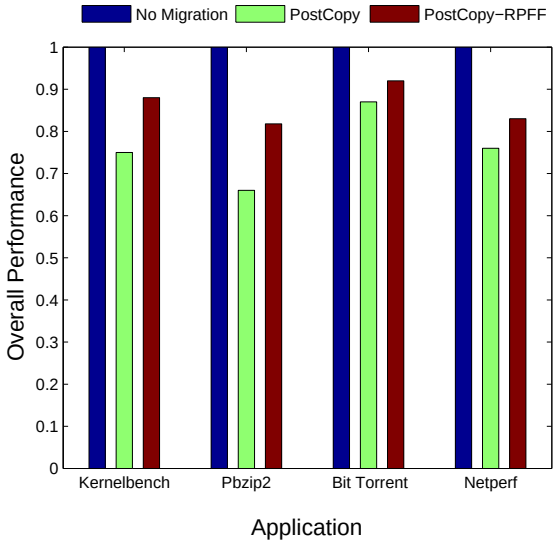

Figure 6: Overall performance for applications during live migration

which bring a non-trivial network delay. As a result, the less remote page faults, the shorter total migration time.

D. Downtime

Downtime is the period during which the service is unavailable due to there being no currently executing instance of the VM; this period will be directly visible to clients of the VM as service interruption. It is a significant performance metric for live migration. As shown in Figure 5, downtime for applications in original post-copy is almost the same as in post-copy with RPFF. This is because that RPFF starts to work after the VM resumed in the destination host, it does not affect downtime at all.

E. Application Performance

The last important measure is performance degradation of the VM caused by live migration. For write-intensive applications in post-copy live migration, remote page faults obviously degrade the application performance due to network latency. We measure the overall performance for the tested applications in post-copy with and without RPFF, as shown in Figure 6, RPFF improve the application performance by 13%,15%, 5% and 7%, respectively.

V. Conclusion and Future Work

In this work , we have proposed RPFF, a remote page-fault filter for post-copy live migration. In post-copy live migration With RPFF, remote page requests caused by page overwriting operations are redirected to local memory instead of fetching the pages from the source host over the network. The experimental results demonstrate that RPFF effectively improve the performance of post-copy live migration.

In the future work, we plan to test more prevalent write-intensive applications to validate the effectiveness of RPFF. To further improve the post-copy live migration, we are going to design a prediction system on the source host to predict memory pages which will be overwritten in the future. Such memory pages can be avoided transferring to the target host and further improve the performance of live migration.

Acknowledgment

This work is supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China under grant 2013ZX03003010-002.

References

[1] 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," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[2] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. Martins, A. V. Anderson, S. M. Bennett, A. Kägi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[4] M. Nelson, B.-H. Lim, G. Hutchins *et al.*, "Fast transparent migration for virtual machines." in *USENIX Annual Technical Conference, General Track*, 2005, pp. 391–394.

[5] 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 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 51–60.

[6] M. E. Russinovich and D. A. Solomon, *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer)*. Redmond, WA, USA: Microsoft Press, 2004.

[7] F. J. T. Fábrega, F. Javier, and J. D. Guttman, "Copy on write," 1995.

[8] "Jonathan corbet. memory compaction. http://lwn.net/articles/368869/, 2010."

[9] M. Gorman and A. Whitcroft, "Supporting the allocation of large contiguous regions of memory," in *Ottawa Linux Symposium (OLS)*, 2007, pp. 141–152.

[10] J. Navarro, S. Iyer, P. Druschel, and A. Cox, "Practical, transparent operating system support for superpages," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 89–104, 2002.

[11] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[12] "Vmware vmotion: Live migration of virtual machines without service interruption. http://www.vmware.com/files/pdf/vmware-vmotion-ds-en.pdf, 2009. online; accessed february 2013."

[13] "Oracle. virtualbox. https://www.virtualbox.org, 2012. online; accessed february 2013."

[14] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High performance virtual machine migration with rdma over modern interconnects," in *Cluster Computing, 2007 IEEE International Conference on*. IEEE, 2007, pp. 11–20.

[15] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive consolidation of virtual machines enabled by postcopy live migration," in *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*. ACM, 2011, pp. 11–18.

[16] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*. ACM, 2009, pp. 101–110.

[17] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–10.

[18] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," *ACM Sigplan Notices*, vol. 46, no. 7, pp. 111–120, 2011.

[19] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proceedings of the 20th international symposium on High performance distributed computing*. ACM, 2011, pp. 135–146.

[20] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, "Jettison: efficient idle desktop consolidation with partial vm migration," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 211–224.

[21] C. Jo, E. Gustafsson, J. Son, and B. Egger, "Efficient live migration of virtual machines using shared storage," in *ACM SIGPLAN Notices*, vol. 48, no. 7. ACM, 2013, pp. 41–50.

[22] N. Amit, D. Tsafrir, and A. Schuster, "Vswapper: A memory swapper for virtualized environments," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 349–366, 2014.

[23] M. Schwidefsky, H. Franke, R. Mansell, H. Raj, D. Osisek, and J. Choi, "Collaborative memory management in hosted linux environments," in *Proceedings of the Linux Symposium*, vol. 2, 2006.

[24] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Enabling instantaneous relocation of virtual machines with a lightweight vmm extension," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 73–83.