Group-Based Memory Deduplication for Virtualized Clouds[⋆]

Sangwook Kim[1], Hwanju Kim[2], and Joonwon Lee[1]

[1] Sungkyunkwan University, Suwon, Gyeonggi-do, Korea
[2] Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea
swkim@csl.skku.edu, joonwon@skku.edu,
hjukim@calab.kaist.ac.kr

Abstract. In virtualized clouds, machine memory is known as a resource that primarily limits consolidation level due to the expensive cost of hardware extension and power consumption. To address this limitation, various memory deduplication techniques have been proposed to increase available machine memory by eliminating memory redundancy. Existing memory deduplication techniques, however, lack isolation support, which is a crucial factor of cloud quality of service and trustworthiness. This paper presents a group-based memory deduplication scheme that ensures isolation between customer groups colocated in a physical machine. In addition to isolation support, our scheme enables per-group customization of memory deduplication according to each group's memory demand and workload characteristic.

Keywords: Memory deduplication, Isolation, Cloud computing

1 Introduction

Intrinsic trade-off between efficient resource utilization and performance isolation arises in cloud computing environments where various services are provided based on a shared pool of computing resources. For high resource utilization, cloud providers typically service a virtual machine (VM) as an isolated component and enable multiple VMs to share underlying physical resources. Although aggressive resource sharing among customers gives a provider more profit, performance interference from the sharing could degrade quality of service customers expect. Performance isolation that ensures quality of service makes it difficult for providers to increase VM consolidation level. Many researchers have addressed this conflicting goal focusing on several sharable resources [2, 4, 5, 7, 12].

Among those sharable resources, machine memory is known as a resource that primarily inhibits high degree of consolidation due to the expensive cost of hardware extension and power consumption [6]. In order to deal with the memory space restriction, memory deduplication has drawn traction as a way of increasing available memory by eliminating redundant memory. Since the memory deduplication was introduced by the VMware ESX server [15], it has

⋆ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0000371).

been well-studied on how effectively to find redundant memory and how to take advantage of saved memory [6, 11]. Due to its effectiveness in reducing memory footprint for hosting requested VM instances, memory deduplication has been appealing to cloud providers who aim to save the total cost of ownership.

Existing memory deduplication techniques, however, lack the functionality of performance isolation in spite of their efficiency. The problem stems from the system-wide operation of memory deduplication across all VMs that reside in a physical machine. In virtualized clouds, a physical machine can accommodate several VMs that belongs to different customers who do not want their sensitive memory contents to be shared with other customers' VMs. Existing schemes do not provide a knob to confine the deduplication process to a group of VMs that want to share their memory one another (e.g., VMs in the same customer or cooperative customers). In addition, the resource usage for system-wide deduplication cannot be properly accounted to corresponding VMs that are involved in sharing. Since resource usage for memory deduplication itself is nontrivial [11, 8], appropriate accounting for the expense of deduplication is a requisite support for cloud computing, which typically employs pay-per-use model.

This paper proposes a group-based memory deduplication scheme that allows the hypervisor to run multiple deduplication threads, each of which is in charge of its designated group. Our scheme provides an interface for a group of VMs, which want to share their memory, to be managed by a dedicated deduplication thread. The group-based memory deduplication has the following advantages. Firstly, memory contents of one group are securely protected from another group. This feature prevents security breaches that exploit memory deduplication [14]. Secondly, the resource usage of deduplication is properly accounted to a corresponding group. Thirdly, a deduplication thread can be customized based on the characteristics and demands of its group. For example, deduplication rates (i.e., scanning rates) can be differently set for each group based on workloads. Finally, memory pages that are reclaimed by a per-group deduplication thread can be readily redistributed to their corresponding group.

The rest of this paper is organized as follows: Section 2 describes the background and motivation behind this work. Section 3 explains the design and implementation of the group-based memory deduplication. Then, Sect. 4 shows experimental results and Sect. 5 discusses issues arising in our scheme and further improvement. Finally, Sect. 6 presents our conclusion and future work.

2 Background and Motivation

2.1 Memory Deduplication

Memory deduplication is a well-known technique that condenses physical memory space by eliminating redundant data loaded in memory. In VM-based consolidated environments, considerable amount of memory can be duplicated across VMs especially when they have homogeneous software stacks such as OSes and applications or work on common working set on a shared storage. By reclaiming redundant memory, the hypervisor can give more memory to a VM whose working set exceeds its memory in order to improve performance. In addition, increase in available memory allows more VMs to run in a physical machine, thereby increasing consolidation level. One representative scheme of memory deduplication is a transparent content-based page sharing [15], which was firstly introduced by the VMware ESX server. This scheme periodically scans physical memory, compares scanned pages based on their contents, merges them if they are identical, and reclaims redundant memory. In order to ensure transparency, a shared page is marked as copy-on-write, by which the shared page will be broken to private copies in response to a write attempt to it.

2.2 Performance Isolation in Clouds

Cloud computing is an emerging technology trend from the perspective of elastic and utility computing on a large shared pool of resources. Among various types of cloud computing, Infrastructure-as-a-Service (IaaS) platform provides a customer with the entire control of software stack in the form of a VM. Provisioned VMs could share the resources of a physical machine according to their service level agreement (SLA). Transparently enabling multiple VMs to share physical resources, cloud providers reduce the number of machines that host requested VM instances, thereby saving the total cost of ownership.

Despite the cost saving, sharing cloud resources intrinsically causes performance interference between individual VMs. Since cloud computing typically complies with a pay-per-use model, the performance a customer expects should not be interfered by other customers' instances. Many researchers have emphasized that sharable hardware resources such as last-level CPU caches [12], machine memory [4], and even entire components of hardware [7], should be properly isolated from each VM.

2.3 Limitations of Memory Deduplication in Clouds

Although memory deduplication improves the performance and consolidation level by exploiting saved memory, existing schemes lack the functionality that ensures isolation among customer instances. The memory deduplication process of the current techniques is globally conducted by the hypervisor. This system-wide memory deduplication poses several issues on performance isolation and trustworthiness.

Firstly, memory contents that come from different customer VMs can be shared. This type of sharing across customer boundary may be unwanted because memory contents could contain sensitive information. In fact, attacks that exploit security breaches of memory deduplication were addressed [11, 14]. Secondly, computational overheads for memory deduplication are not properly accounted to each corresponding customer. Memory deduplication entails computational tasks including scanning, hashing, byte-by-byte comparison, and copy-on-write breaking. Since these tasks are done in a single execution context, their CPU usages cannot be billed to an appropriate VM whose memory is involved in deduplication. Thirdly, a deduplication rate should be globally set without considering customers' demands or workload characteristics. The pace at which identical pages are shared determines a reprovisioning rate of reclaimed memory, which contributes to performance improvement.

3 Group-based Memory Deduplication

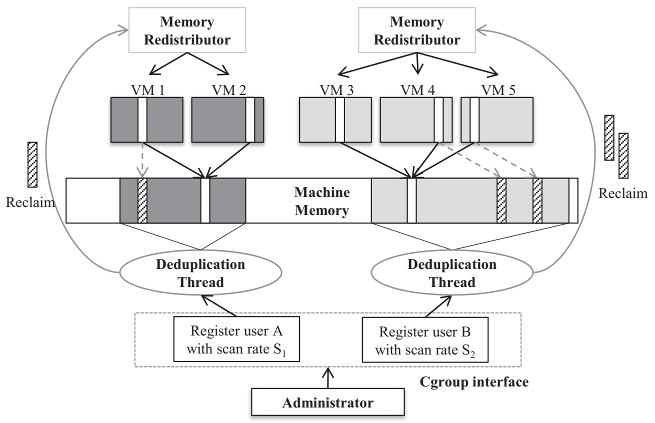


Fig. 1. Architecture overview of the group-based memory deduplication

3.1 Design

Our group-based memory deduplication scheme provides a mechanism that supports multiple deduplication threads, each of which is dedicated to each group defined by administrators. The interface for grouping VMs is exposed to user space so that administrators can readily adjust grouping policies and per-group customization on the fly. Figure 1 illustrates the architecture overview of our mechanism.

In order to ensure isolation between groups, a deduplication thread is involved in virtual address spaces that are registered to its group. Accordingly, all deduplication operations are solely carried out on a designated memory space of each group. Since a deduplication thread is bound to its group, the resource usage for deduplication can simply be accounted to its group. Besides deduplication, redistributing reclaimed memory is done within each group. As shown in Fig. 1, a per-group deduplication thread notifies its corresponding memory redistributor of how many pages are reclaimed by its group. Each per-group memory redistributor supplies the reclaimed memory to its group so that the VMs of the group take advantage of increased memory.

In addition, an administrator can differently set scan rates according to each group's demand or workload characteristics. For example, a high scan rate can be set to a group if its customer wants aggressive scanning in favor of additional memory, by which performance benefits outweigh scanning overheads. Conversely, a group that has CPU-intensive workloads with enough memory may desire a low scan rate. This per-group scan rate gives more flexibility by allowing group-specific customization.

3.2 Implementation

We implemented the prototype of our scheme by extending the Linux KSM [1]. The current KSM conducts system-wide memory deduplication over virtual address spaces that are registered via the *madvise* system call. When Kernel Virtual Machine (KVM) [9] creates a VM instance, it automatically registers the VM's entire memory regions to KSM. Once KSM is initiated, a global deduplication thread, named *ksmd*, performs deduplication with respect to all VM's memory regions. For group-based memory deduplication, we modified this system-wide deduplication algorithm by splitting the global ksmd into per-group ksmds. Each per-group ksmd operates with its own data structures that are completely isolated from other ksmds.

For a grouping interface, we used the *cgroup* [10], which is a general component to group threads via the Linux VFS. We added the *KSM cgroup subsystem* for administrators or user applications to easily define deduplication groups. Each group directory includes several logical files, which indicate a scan rate and the number of shared pages to interact with its per-group ksmd.

Taking advantage of the cgroup interface, the memory redistributor is simply implemented as a user-level script. This script periodically checks the number of reclaimed pages for each group and reprovisions them to a corresponding group by interacting with a guest-side balloon driver. Regarding intra-group reprovisioning, our current version evenly supplies given memory to VMs within a group. However, more sophisticated policies can be applied by using working set estimation techniques.

4 Evaluation

In this section, we present preliminary evaluation results to show how the group-based memory deduplication scheme impacts on memory sharing and redistribution behaviors.

4.1 Experimental Environments

Our prototype is installed on a machine with Intel i5 quad core CPU 760 2.80GHz, 4GB of RAM, and two 1TB HDDs. This host machine runs Ubuntu 10.10 with the *qemu-kvm* 0.14.0 and our modified Linux kernel 2.6.36.2. We compared our scheme, called *GRP*, with two baseline schemes: *NOGRP-equal* and *NOGRP-SE*. While the two baselines have non-group memory deduplication in common, they have different reprovisioning policies. NOGRP-equal reprovisions reclaimed memory evenly to existing VMs, whereas NOGRP-SE gives a VM reclaimed memory in proportion to its *sharing entitlement*, which means how much contribution a VM makes to save memory; this reprovisioning scheme was proposed by Milos et al [11]. For example, if two VMs make all reclaimed pages, they deserve to receive all additional memory they contribute. From the perspective of isolation, we believe that this scheme is more suitable than the equal reprovisioning for cloud environments.

We evaluated a two-group scenario where each group has two VMs configured as follows:

- **MR group** includes two VMs that run a distributed *wordcount* on the Hadoop MapReduce framework. Hadoop slave instances concurrently compute with a 200MB input file in the two VMs, one of which is also in charge of the master for controlling the slaves. This group uses Ubuntu 10.10 as a guest OS.
- **FIO group** includes two VMs each of which run a random read workload on 700MB common data set. We used *sysbench* and measured average throughput for 400 seconds. This group uses Fedora 14 as a guest OS.

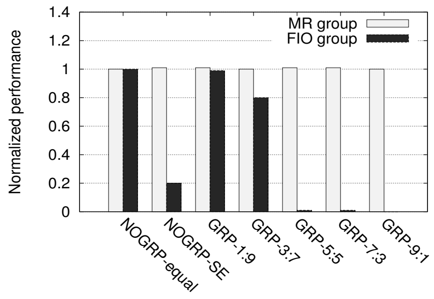
To minimize interferences between groups, we used *cpu*, *cpuset*, and *blkio* cgroup subsystems for both NOGRP and GRP. NOGRP baselines allow a global ksmd to belong to its own group, while our scheme makes each per-group ksmd belong to its corresponding group so that deduplication cost is accounted to its group. The groups of main workloads including ksmd group (NOGRP case) has sufficiently higher CPU shares than other system threads in order to minimize the effect of system daemon activities.

4.2 Effects of Group-based Memory Deduplication

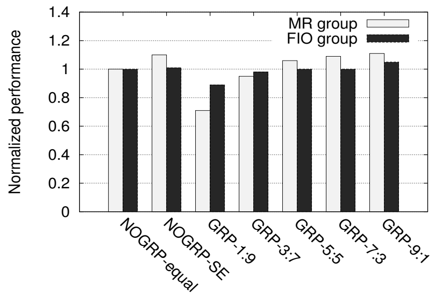
We evaluated the performance and memory changes with sharing trends for two configurations, in which one group has enough memory to cover working set while the other does not. FIO_{low} indicates that the FIO group does not have enough memory to cover its working set (MR-VM:FIO-VM=640MB:640MB), whereas MR_{low} indicates that the MR group lacks memory for its working set (MR-VM:FIO-VM=384MB:896MB). With respect to our scheme, we varied scan rates for each group; GRP-x:y means the ratio of scan rates for MR and FIO. To compare the performance across all policies, we make the sum of scan rates for each policy equal (10,000 pages/sec).

Figure 2 shows the normalized throughput of each group for different policies. The first thing to note is two NOGRPs show different performance. In the case of FIO_{low}, the FIO group of NOGRP-equal shows much higher performance than that of NOGRP-SE. To investigate this difference, Fig. 3 shows the changes in memory for each VM with the amount of reclaimed memory as time progresses. For both cases, the MR group emits a large amount of reclaimed memory for 25–60 seconds. Although the MR group has the contribution for the reclaimed pages during the period, NOGRP-equal reprovisions them evenly to the two groups. Since the FIO group lacks the memory in FIO_{low}, such aid of additional memory boosts its performance. Furthermore, the increased memory helps the FIO group make more reclaimed memory by sharing more pages. On the other hand, NOGRP-SE reprovisions the initial reclaimed memory to only the MR group based on its sharing entitlement, so that the FIO group cannot benefit from any additional memory during the initial period.

Conversely, in the case of MR_{low}, the MR group of NOGRP-SE achieves higher performance than that of NOGRP-equal. As shown in Fig. 4, NOGRP-SE makes the MR group quickly receive more memory contributed by its own

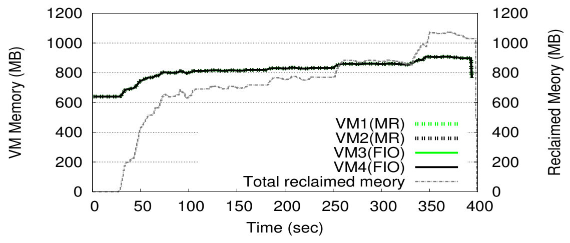


(a) FIO_{low}

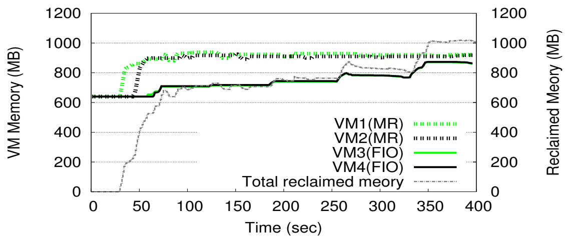


(b) MR_{low}

Fig. 2. Normalized performance for NOGRP-equal, NOGRP-SE, and GRP with various scan rates (x:y is the scan rates of MR:FIO).

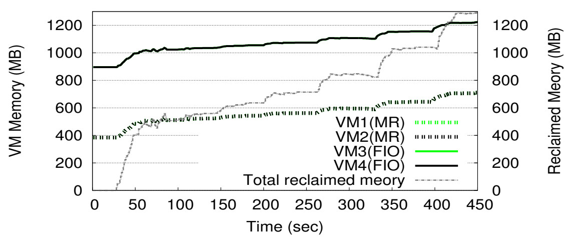


(a) NOGRP-equal

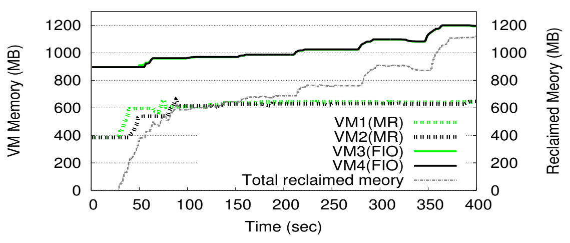


(b) NOGRP-SE

Fig. 3. Memory changes in the NOGRP cases with reclaimed memory (FIO_{low})



(a) NOGRP-equal

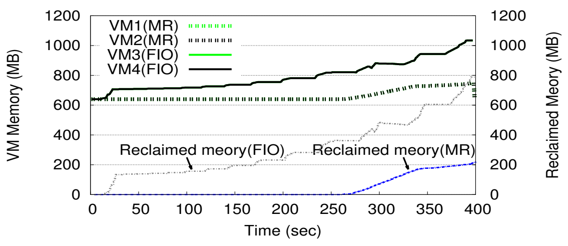


(b) NOGRP-SE

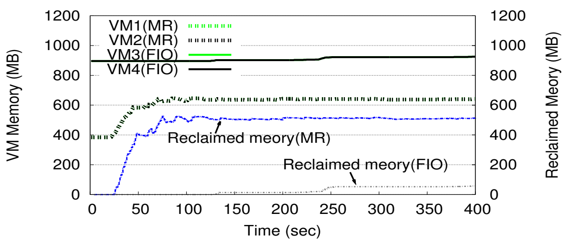
Fig. 4. Memory changes in the NOGRP cases with reclaimed memory (MR_{low})

sharing during the initial period, thereby boosting the performance of the MR group. The results of FIO_{low} and MR_{low} imply that neither of the non-group schemes (NOGRP-equal and NOGRP-SE) always achieves the best performance, since each group's memory demands are different.

Figure 2 also shows the results of the group-based memory dedupication with various scan rate settings. As shown in the figure, the best performance results are achieved on certain scan rate ratios: 1:9 for FIO_{low} and 9:1 for MR_{low}. It is intuitive that a higher scan rate makes a group that lacks memory quickly reap additional memory, thereby improving its performance. Figure 5 shows the two cases of the best performance. As expected, a high scan rate quickly produces reclaimed memory, which is then reprovisioned to a group that desires more



(a) GRP-1:9 (FIO_{low})



(b) GRP-9:1 (MR_{low})

Fig. 5. Memory changes in the best performance cases of GRP with reclaimed memory

memory. Although a low scan rate slowly emits a small amount of reclaimed memory, the performance of a group that has enough memory is not affected.

As a result, the group-based memory deduplication can achieve the best performance if a scan rate for each group is appropriately chosen. Considering that NOGRP-SE is currently the most suitable approach for clouds, due to its capitalism, it does not have room for customization on the basis of each group's memory demand and workload characteristic. In Sect. 5.3, we discuss our plan to devise the dynamic adjustment of per-group scan rates.

5 Discussion

In this section, we discuss promising applicability of the group-based deduplication focusing on VM colocation, various grouping policies, and feasible customization of per-group deduplication.

5.1 VM Colocation

For the group-based memory deduplication to be effective, multiple VMs within the same group should be colocated in a physical machine. Assuming that a group is established based on a customer, there are several cases to colocate VMs from the same customer. Firstly, as novel hardware (e.g., many core processors and SR-IOV network cards) has been increasingly supporting consolidation scalability [7], a physical machine becomes capable of colocating the increasing number of VMs. This trend increases the likelihood that VMs from the same customer are colocated. Secondly, VM colocation policies that favor cloud-wide resource efficiency (e.g., memory footprint [16] and network bandwidth [13]) would encourage a cloud provider to colocate VMs from the same customer. For example, if a cloud customer leases VMs for distributed computing on the MapReduce framework, the VMs have homogeneous software stack, common working set, and much communication traffic among them. In this case, a cloud provider seeks to colocate such VMs in a physical machine for efficiency as long as their SLAs are satisfied.

Although the same customer's VMs are not colocated, there are still chances to take advantage of the group-based memory deduplication. As cloud computing has been embracing various services, there are growing opportunities to share data among related services. CloudViews [3] presents a blueprint of rich data sharing among cloud-based Web services. We expect that such direction allows our scheme to group cooperative customers who agree with data sharing. In addition, intra-VM memory deduplication may not be negligible depending on workloads when a VM is solely located in a group. Some scientific workloads have a considerable amount of duplicate pages in native environments [1].

5.2 Grouping Policies

We are currently considering various grouping policies other than the customer-based isolation policy. Intuitively, VMs can be grouped based on their sharing opportunities likely attained by the common software stack and working set [15]. To this end, we can statically group the same virtual appliances or distributed computing nodes. For dynamic grouping, a cloud provider can figure out sharing opportunities by keeping track of memory fingerprint on the fly. In the case of clouds, which do not allow arbitrary grouping across independent customers, providers can offer their customers a grouping option that benefits from more available memory by sharing in a symbiotic manner.

Similarly, a cloud provider can service a pricing model that offers best-effort available memory with the lower bound guarantee. Note that the additional memory reprovisioned via deduplication can be returned by copy-on-write breaking at any time. For this reason, such additional memory is provided to customers in a best-effort manner. The group-based memory deduplication can group VMs that participate in this type of memory provisioning. Nathuji et al. [12] proposed this type of pricing model with respect to CPU capacity offering.

5.3 Per-group Deduplication Customization

The group-based memory deduplication enables per-group customization for deduplication process. As shown in Sect. 4, the performance of applications that require more memory for covering their working set relies on memory reprovisioning rates. Based on the results, we are extending our scheme to support dynamic deduplication rates by monitoring workloads for each group. Currently, we take two metrics into account for scanning rate adjustment.

Firstly, the hypervisor can monitor how many pages are being reclaimed for each group during a certain time window. When VMs in a group abruptly start loading a large amount of identical pages, the number of pages shared will rapidly increase. In this case, a higher scanning rate boosts the reprovisioning rate of additional memory, thereby improving the performance. Secondly, when workloads in a group become CPU-intensive, a high deduplication rate may degrade their performance due to deduplication overheads. Since the deduplication process may pollute CPU caches and consume memory bandwidth, these overheads may offset or outweigh the benefits of deduplication with regard to CPU-intensive workloads. In this case, it is important to determine an appropriate rate for overall performance by considering CPU usage and memory demands.

6 Conclusions and Future Work

In this paper, we devise a knob to group VMs that allow their memory to be shared one another. The proposed scheme enables the memory deduplication process to be isolated between groups and customized based on each group's demand and characteristic. We believe that the group-based isolation is an essential feature of memory deduplication in cloud computing environments, which regard performance isolation and trustworthiness as crucial factors.

As discussed, we plan to explore various grouping policies and dynamic adjustment of deduplication rates on the basis of workload characteristics. Furthermore, we are investigating a flexible reprovisioning scheme that effectively exploits reclaimed memory to improve overall performance in the same group.

References

1. A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In *Proc. OLS*, 2009.
2. T. Cucinotta, D. Giani, D. Faggioli, and F. Checconi. Providing performance guarantees to virtual machines using real-time scheduling. In *Proc. VHPC*, 2010.
3. R. Geambasu, S. D. Gribble, and H. M. Levy. Cloudviews: Communal data sharing in public clouds. In *Proc. HotCloud*, 2009.
4. A. Gordon, M. R. Hines, D. da Silva, M. Ben-Yehuda, M. Silva, and G. Lizarraga. Ginkgo: Automated, application-driven memory overcommitment for cloud computing. In *Proc. RESoLVE*, 2011.
5. D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proc. Middleware*, 2006.
6. D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *Proc. OSDI*, 2008.
7. E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: Virtualized cloud infrastructure without the virtualization. In *Proc. ISCA*, 2010.
8. H. Kim, H. Jo, and J. Lee. XHive: Efficient cooperative caching for virtual machines. *IEEE Transactions on Computers*, 60(1):106–119, 2011.
9. A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: The Linux virtual machine monitor. In *Proc. OLS*, 2007.
10. P. B. Menage. Adding generic process containers to the Linux kernel. In *Proc. OLS*, 2007.
11. G. Miłós, D. G. Murray, S. Hand, and M. A. Fetterman. Satori: Enlightened page sharing. In *Proc. USENIX ATC*, 2009.
12. R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proc. EuroSys*, 2010.
13. J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Proc. ICPP*, 2010.
14. K. Suzaki, K. Iijima, T. Yagi, and C. Artho. Memory deduplication as a threat to the guest OS. In *Proc. EuroSec*, 2011.
15. C. A. Waldspurger. Memory resource management in VMware ESX server. In *Proc. OSDI*, 2002.
16. T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner. Memory buddies: Exploiting page sharing for smart colocation in virtulized data centers. In *Proc. VEE*, 2009.