ESX Memory Resource Management: Swap

Ishan Banerjee Philip Moltmann Kiran Tati Rajesh Venkatasubramanian
VMware Inc
{ishan, moltmann, ktati, vrajesh}@vmware.com

Abstract

As virtualization matures into a mainstream datacenter component, there is an increasing demand for improving consolidation ratios. *Consolidation ratio* measures the amount of virtual hardware placed on physical hardware. A higher consolidation ratio typically implies greater operational efficiency in the datacenter.

VMware® ESX® is a reliable and efficnent hypervisor, enabling high consolidation ratios, reliable operation and efficient utilization of hardware resources. The memory resource management (RM) system of ESX distributes hardware memory resources to virtual machines (VMs) in a fair and efficient manner. It provides certain resource management guarantees while ensuring the stability and reliability of ESX and powered-on VMs.

To enable high consolidation ratios in a reliable manner, the memory RM system enables overcommitting of ESX. When VMs, whose total configured memory size exceeds the memory available to ESX are powered on, ESX is considered to be memory-overcommitted. The memory RM system balances memory distribution between powered-on VMs dynamically when ESX is memory-overcommitted as well as under-committed. It utilizes the hypervisor-level swap system to reclaim unused memory from VMs.

The swap system is the backbone for enabling reliable memory overcommitment. It is an integral part of the memory RM system. It is integrated with VMware vMotion technology to enable seamless and efficient migration of VMs from one host to another.

This article describes the swap system. It shows how swap spaces are created, configured and managed on ESX.

General Terms memory management, memory reclamation, memory swapping

Keywords ESX, memory resource management

1. Introduction

VMware ESX enables reliable, efficient operation of virtual machines (VM) in a datacenter. The key enablers for achieving these are the VMware overcommitment technology and vMotion technology. ESX enables memory overcommitment and CPU overcommitment.

The VMware memory-overcommitment technology enables a user to power on VMs, whose total configured memory size (virtual RAM or vRAM) exceeds the total physical memory (pRAM) available to ESX. Memory overcommitment does not mean that ESX will magically provide powered-on VMs with more memory than that available to ESX. It provides memory to those VMs that need it most and reclaims memory from those VMs that are not actively using memory.

This article appears in the VMware Technical Journal (VMTJ) Vol. 3, No. 1–June 2014 issue.

Memory reclamation is therefore a part of memory overcommitment. Memory reclamation reclaims memory from VMs that are not *actively* using it. The reclaimed memory is then given to a VM that needs it to perform better. At all times, the total memory given to all powered-on VMs remains less than or equal to the memory available to ESX.

In order to enable VMs to perform at their best, ESX must carefully determine which VM to reclaim memory from and which VM to give memory to. This decision is made by the memory management policy-maker of ESX, termed *MemSched*. Each powered-on VM with configured memory size, *memsize*, has certain memory-attributes attached to it. They are *memory reservation*, *memory limit* and *memory shares*. These parameters, termed RLS parameters, are used by MemSched along with certain other dynamic attributes of the powered-on VM to decide on the memory that needs to be reclaimed from (or given to) the VM. Detailed description of the memory management policies implemented in MemSched is beyond the scope of this article.

MemSched decides on the memory that must be reclaimed from a VM. It then instructs the reclamation components of ESX to reclaim the targeted memory from the VM. The reclamation components that reclaim memory from a VM are – the transparent page sharing component, balloon component, compression component and hypervisor-level swap component [7]. The reclamation components reclaim memory from VMs by using their respective methods and release the reclaimed memory to ESX. ESX can then allocate this memory to VMs that need it most.

The page sharing component attempts to reclaim memory from a VM when the contents of a memory page fully match the contents of another memory page in the same or a different powered-on VM on that ESX server. The two pages are collapsed into one page. Both VMs contain a reference to the single page and access it in a read-only manner. Thus, when two pages are shared into one page, one memory page is considered reclaimed. Similarly, many memory pages with the same content can refer to one read-only page with the same content. None of the VMs is aware that it is actually sharing a memory page with another VM. Sharing is completely transparent to the VMs.

ESX shares memory pages at a granularity of 4KB only. The page sharing component is active at all times. It continuously scans VMs for shareable pages and shares them whenever possible. A VM with many shared pages consumes fewer memory pages than the total amount of its configured memory space that has been mapped by the guest OS. The actual amount of savings depends on the number of physical pages backing the virtual memory pages and can be determined at run-time only. A VM with many shared pages exerts less memory pressure on ESX than one that does not have many shared pages, because it consumes less memory.

The balloon component of ESX interacts with the ESX balloon driver inside the guest OS. The balloon component reclaims memory from a VM by instructing the balloon driver to allocate guest memory pages and pin them in the guest memory. These pages are

known to ESX as belonging to the balloon driver. ESX can then allocate those pages to other VMs. This approach effectively utilizes the guest OS's memory reclamation logic to swap-out cold memory pages inside the guest OS into the guest OS's swap space.

The compression and swap stages operate in sequence. The compression component compresses memory pages before they are swapped out by the hypervisor-level swap component of ESX. If the compression ratio is good, then the page is retained as a compressed page. If the compression ratio is not good, then the page is swapped out to the hypervisor-level swap space of ESX. Compressing a page yields less than one page's worth of effective reclamation. Swapping a page yields one reclaimed page.

These four memory reclamation techniques are employed to reclaim guest memory from a VM when required. This article describes the hypervisor-level swap component of ESX in vSphere 5.5. The rest of this article is organized as follows. Section 2 describes the memory reclamation mechanism of ESX, Sections 3 and 4 describe the swap component of ESX in the context of VMs, Section 5 describes the swap component in the context of user-worlds (UWs) and Section 7 completes this article with a closing discussion.

2. Background and related work

Efficient use of main memory continues to be a priority for OSs and hypervisors even though the cost of main memory continues to decrease. In this section recent work related to memory reclamation in hypervisor are described. The necessity of memory reclamation in ESX is also shown.

Related work: Contemporary OSs and hypervisors employ one or more mechanisms to reclaim memory from their memory consumers. Content-based page sharing is used in Linux/KVM and ESX, memory paging is used by most OSs and hypervisors (ESX, KVM, Hyper-V, Xen) and memory ballooning is used by ESX and Xen. The research community has also demonstrated memory reclamation techniques in Singleton [6], Satori [4], Difference Engine [2], KSM [1], CMM [5] and Ginkgo [3].

Background: MemSched is the ESX component that determines the memory to be given to each powered-on VM. It employs the fours reclamation techniques – transparent page sharing, ballooning, compression and hypervisor-level swapping – to reclaim memory from powered-on VMs when required.

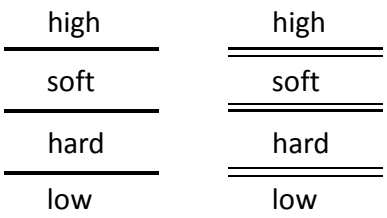
When a powered-on VM allocates memory, ESX does not prevent it from doing so. However, if a VM consumes more memory than its fair share, then ESX (via MemSched) reclaims the amount in excess of its fair share. ESX has four operational memory states. These are – `high`, `soft`, `hard` and `low`. These states are determined by the free memory available for allocation at a given time compared to the total memory of ESX. Table 1 shows a schematic representation of the memory states of ESX. ESX attempts to stay in the `high` state at all times. Whenever it dips into any of the other states, ESX reclaims memory from VMs until it enters the `high` state again. The values of the state threshold depend on the ESX memory. Each threshold consists of two sub-thresholds, not exposed to the user. This prevents oscillation of ESX memory state near each threshold.

The memory allocation and reclamation behavior of ESX depends on the state of ESX. Table 2 lists the actions ESX initiates at different states.

In the `high` memory state, ESX attempts the page sharing reclamation technique only. Page sharing works on *small* – 4KB sized – pages only. If a VM contains *large* – 2MB sized – pages, then the page sharing technique ignores them for the time being. This is a passive reclamation mode where ESX opportunistically attempts to reclaim memory from VMs without affecting their performance.

In the `soft` memory state, ESX attempts to actively reclaim memory from VMs. It computes and sets a balloon target for VMs that might have consumed more than its share of memory. The balloon drivers in those VMs expand, releasing memory to ESX. This process continues until ESX enters the `high` state. ESX is dependent on the balloon driver and the guest OS to reclaim memory using the ballooning technique. If the balloon driver is not running or the guest OS is not responsive or the guest OS is unable to allocate memory to the balloon driver, then no memory can be reclaimed using this technique. Ballooning takes place in addition to page sharing.

If memory reclamation using ballooning is not fast enough and VMs continue to allocate memory, then ESX can enter the



(a) (b)

Table 1. Free memory state transition threshold in ESX. (a) User visible (b) Internal threshold to avoid oscillation.

State	pshare	balloon	compress	swap	block
high	✓				
soft	✓	✓			
hard	✓		✓	✓	
low	✓		✓	✓	✓

Table 2. Actions performed by ESx in different memory states of ESX.

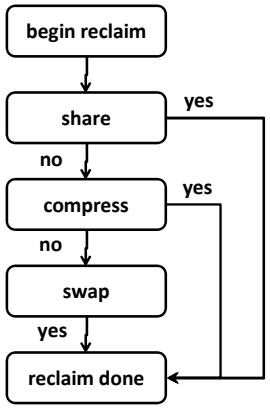


Figure 1. Steps performed by ESX when swapping out guest memory page.

`hard` memory state. In this state, ESX attempts to actively and aggressively reclaim memory from VMs. ESX computes and sets a swap target for VMs that are consuming memory in excess of their fair share. The swap component of ESX attempts to swap out the targeted memory pages from those VMs.

Figure 1 shows the actions performed on a guest memory page while swapping it out to the swap space.

For each guest memory page selected for swapping out, ESX opportunistically attempts to share it. If the page is shareable, it is shared, instead of being swapped out. If the page is not shareable, ESX now attempts to compress the page. If compressed with a good compression ratio, then the page does not need to be swapped out. If the page is not compressible, then ESX swaps the page out to its swap space. If a VM contains a large page – 2MB size – ESX converts the page into small – 4KB size – pages before attempting to share or compress it. Therefore, if a VM contains many large pages, that might not have been shared earlier, they will undergo an attempt at sharing at this time.

If VMs allocate memory at a rate faster than can be swapped out, ESX can enter the `low` memory state. In this state, memory is critically low and ESX makes every attempt to prevent memory exhaustion. This is typically done by preventing VMs from allocating memory while swapping memory away from VMs. A VM attempting to allocate memory will be blocked from allocation until enough memory has been reclaimed from it.

Among the four memory reclamation techniques – page sharing, ballooning, compression and hypervisor-level swapping – the first three are opportunistic. They depend on factors outside the control of ESX for reclaiming memory from a VM. Page sharing and compression depend on the contents of a guest memory page. Ballooning depends on the balloon driver and the stability of the guest OS. The fourth method, hypervisor-level swap, is the only guaranteed method for reclaiming memory from a VM. In the event that the first three methods are not effective or fast enough, ESX swaps out memory from a VM in order to maintain its stability.

The ESX swap system is therefore designed to guarantee reclamation of memory from VMs under all circumstances. The swap space is created such that it is not exhausted. In addition, memory management policies and mechanisms of ESX ensure that memory can always be successfully reclaimed from VMs when required.

A *swap-out* operation is defined as reclamation of guest memory pages by ESX, using hypervisor-level swapping, into a suitable swap space attached to ESX. A *swap-in* operation is defined as reading a swapped-out memory page from the swap space, when it is accessed by the VM. Swap-out operations are typically asynchronous to the execution of the VM and guest OS. Swap-in operation are always synchronous with the access made by the VM or the guest OS. If a swap-in operation fails owing to storage device failure, then the corresponding VM is terminated immediately.

In the next section, a detailed description of the swap component is presented.

3. Swap without vMotion

ESX is designed to guarantee memory reclamation by swapping out guest memory pages under all conditions. This ensures that ESX never runs out of memory when overcommitted and under memory pressure from guest workloads. ESX can reclaim memory from VMs (see Section 4) as well as native processes, known as user-worlds (UWs) (see Section 5), by hypervisor-level swapping.

This section describes memory reclamation, by swapping, from VMs, in the absence of vMotion.

3.1 Simple virtual machine

ESX creates a swap space for each powered-on VM. Powered-off VMs do not require a swap space. The swap space for a powered-on VM is created in the form of a fully-allocated (also known as *thick*) per-VM swap file. There is one swap file for each powered-on VM. All storage blocks of the file are allocated from the underlying storage space before the VM successfully powers on. The location of this swap space defaults to the same location as the `.vmx` configuration file for the VM. This swap file is used for reclaiming memory from the corresponding VM only.

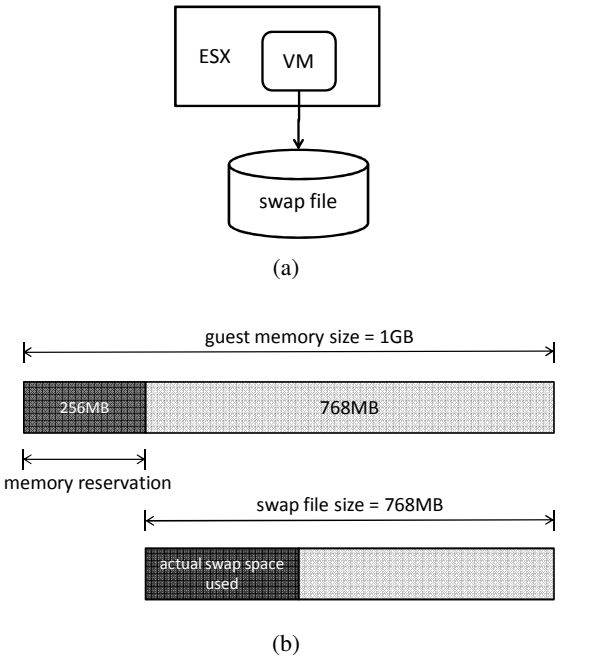


Figure 2. (a) Schematic representation of a VM linked to its per-VM swap space. (b) Swap space consumed by a powered-on VM. The VM has a configured memory size of 1GB and a configured memory reservation of 256MB.

Figure 2 (a) shows the schematic representation of an ESX with one powered-on VM. The VM is linked to its per-VM swap space. This swap space is distinct from the swap space created by the guest OS inside the VM. ESX does not have any control over the swap space created by the guest OS.

Memory reclaimed from this VM by means of swapping is placed in this swap space. Figure 2 (b) shows the swap space created for a single powered-on VM.

The size of the swap space for this VM, at power-on, is given by:

$$swapsz = memsize - memory\ \ reservation \qquad (1)$$

where *memsize* is the configured guest memory size for the VM and *memory reservation* is the corresponding setting for the VM. For the example in Figure 2, a swap space of size $768MB$ is created.

ESX gurantees that memory up to *memory reservation* will always be present to back the virtual memory of the VM. Virtual memory in excess of this amount can be reclaimed from the VM by ESX, if required. The swap space is sized such that there are always enough storage blocks to back the reclaimed memory pages. These storage blocks are allocated in the file system before declaring the VM as powered-on. The actual space consumed from the swap space can be less than its size. This depends on the amount of memory being reclaimed from the VM using swapping.

When the *memory reservation* of a powered-on VM is reduced, the VM's swap space is recalculated and extended, if required, using Equation 1. However, when a powered-on VM's *memory reservation* is raised, the swap space is not reduced. This can result in a swap space that is larger than required. A subsequent reduction in *memory reservation* might or might not result in a change in the swap space.

Typically, when memory is reclaimed by means of hypervisor-level swapping, the execution of the guest OS is not stalled. This means that the guest OS and workload execution are not blocked by the write operation of memory pages to the swap space. However, when the guest OS accesses a memory page that has been swapped out by ESX, ESX handles the access by reading in the swapped page from the swap space. The guest vCPU generating the access waits during this time. This waiting time can adversely affect the guest performance.

Figure 3 shows a schematic representation of a VM whose memory pages have been reclaimed using different techniques. Figure 3 (a) shows the state of the VM before pages were reclaimed by hypervisor-level swapping. In this stage, some memory pages are already shared. The memory saved by sharing is shown as `P`. Some memory has also been reclaimed by ballooning, shown as `B`. In this stage, the memory pages being *consumed* by the VM are more than its *entitlement*. Also, ESX is in the `low` state. This triggers memory reclamation by swapping. In Figure 3 (b), memory has been reclaimed by swapping using the steps in Figure 1. During the swapping process, some pages were found to be compressible and hence compressed, shown as `C`, some pages were found sharable and hence shared. Those pages that could not be compressed or shared were swapped to the per-VM swap space. This is shown as `S`.

The scenario described in this subsection shows one VM powered on in ESX. When ESX needs to reclaim memory from this VM, it will follow the steps shown in Figure 1.

3.2 Virtual machine with Solid State Device

The scenario described in this subsection is similar to the one in sub-section 3.1. In addition, the ESX server on which this VM is powered on has an attached *solid state device* (SSD).

ESX takes advantage of an SSD to enable faster swap operations. The read operations from an SSD are several orders of magnitude faster, than a read operation from a spinning disk. Hence, the swap-in operation of a guest memory page takes place much faster when reclaimed memory is placed on an SSD.

The SSD is a global resource to ESX. ESX can use a part of the SSD to reclaim memory pages from VMs. ESX can be configured to create a global swap space on the SSD. All VMs share this swap space. The amount of storage space that ESX can consume from the SSD as a global swap space is configured by the user.

The size of the swap space created on the SSD is user-defined. It is not related to any attribute of ESX or VMs.

Figure 4 shows a schematic representation of the swap space when ESX uses an SSD. The SSD is used in addition to the per-VM swap space. For a given VM, the SSD is a global shared resource, whereas its per-VM swap space is a private resource. The global shared swap space on the SSD is available for reclaiming memory from a VM on a first-come-first-served basis.

Figure 5 shows the steps when ESX swaps out a memory page from a VM, in the presence of swap space on an SSD. This figure expands on the *swap* step from Figure 1. ESX first attempts to swap out the guest memory page to the global swap space on the SSD. If the swap space on the SSD is full, then ESX swaps out the page to the per-VM swap space.

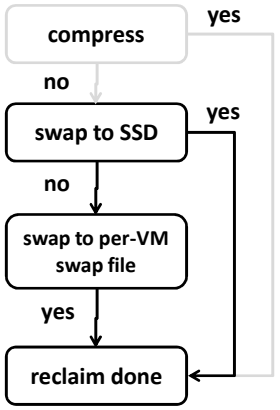


Figure 5. Steps performed by ESX when swapping out a guest memory page when swap space on SSD is configured.

ESX does not guarantee swap space on the SSD to a VM because its size is configured by the user and space from the SSD is allocated to VMs on a first-come-first-served basis. However, swap space is always available on the per-VM swap space. Figure 3 (c) shows the swapped pages being written to the global SSD swap space. In this case there is enough space on the SSD to accommodate all swapped pages for the VM. In Figure 3 (d) the global SSD space has been consumed by other VMs. There is not enough space for the VM in this figure. Hence excess pages overflow into the per-VM swap space. If the swap space on an SSD becomes full, then swapped memory from a VM is written directly into the VM's per-VM swap space. This ensures reliable operation of the VM.

However, it can be noted that memory pages residing on the swap space on the SSD for a long time are probably *cold* memory pages and might not be required by the VM in the near future. The incoming pages to the SSD, which are being swapped out are likely hotter than those already on the SSD.

ESX recognizes this effect. Therefore, when the free space on the SSD swap space falls below a watermark, ESX actively transfers swapped memory pages from the SSD swap space into the per-VM swap space of the corresponding VM. As a result, pages residing on the SSD swap space are typically hotter than those residing on the per-VM swap space. Therefore, if the hot pages are subsequently accessed by the VM, they are more likely to be found on the SSD swap space than on the per-VM swap space. This results in a faster swap-in operation. Figure 6 illustrates the transfer of swapped guest memory pages from the SSD swap space to the per-VM swap space when the SSD swap space is almost full. This step is an optimization. If ESX is not quick enough to transfer pages, then the incoming pages overflow to the per-VM swap space.

The scheme described in Figure 3 (c) is desirable in the presence of memory reclamation by swapping. This is because SSD is typically several orders of magnitude faster than magnetic disk storage. Hence swap read and write operations from SSD are fast. This results in reduced latency when a swapped page is being read in from the SSD swap space, compared to reading it in from the per-VM swap space.

This section described swapping in the absence of vMotion. The next section describes swapping in the presence of vMotion.

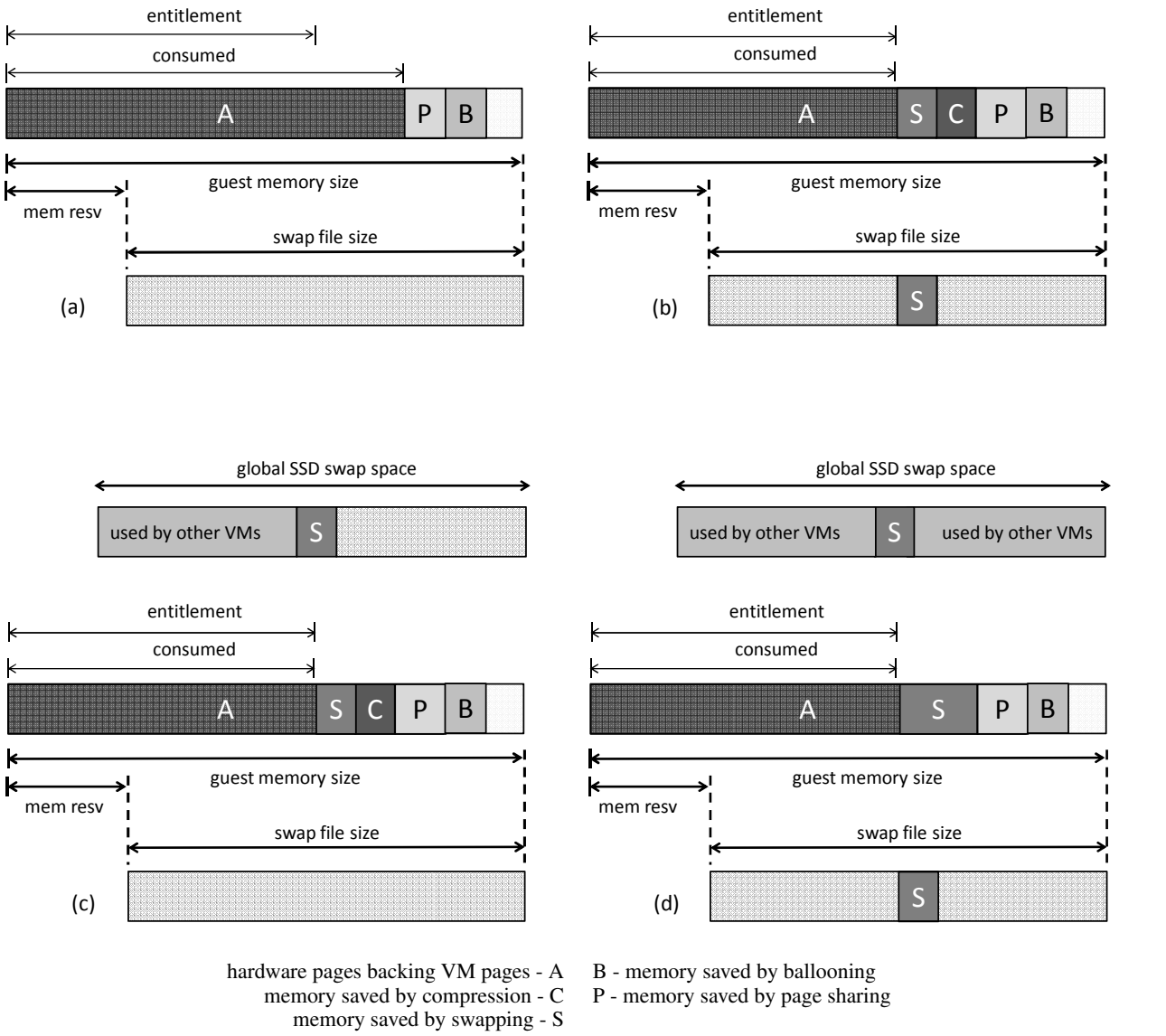


Figure 3. State of a VM before and after reclaiming pages by swapping. a) State of the VM in `low` ESX state before memory reclamation by swapping. b) memory reclaimed by swapping, some more pages have been shared, compressed. c) In the presence of global SSD swap space, swapped pages are written to SSD. d) In the presence of global SSD swap space, pages are written to SSD, and overflow into per-VM swap space.

4. Swap with vMotion

vMotion is VMware's technology that migrates a powered-on VM from one ESX server to another with no downtime. ESX memory resource management technology is integrated with its vMotion technology and provides seamless performance during migration.

ESX supports two types of migration – shared-swap vMotion and unshared-swap vMotion. The swap space is treated differently in these two types of vMotion. They are described in the following subsections. The size of the swap space is given by Equation 1.

4.1 Shared-swap vMotion

Shared-swap vMotion takes place when a powered-on VM migrates from one ESX server to another such that the per-VM swap file of both the source and the destination VM are the same. The per-VM swap file is stored on a shared storage device. It is accessible from the source VM before and during the vMotion. It is subsequently accessed by the destination VM. The presence of a shared storage device between the source and destination ESX server does not make a vMotion a shared-swap vMotion. A vMotion is termed shared-swap vMotion if and only if the swap file being used by the source VM is the same as the one to be subsequently used by the destination VM.

Figure 7 shows the steps during shared-swap vMotion. In these steps, a VM is migrated from a source ESX (left) to a destination ESX (right). The two ESX servers share a common storage device. The swap file of the VM is located on this shared device.

In Figure 7 (a), which is the initial state, the source VM, on the source ESX server is linked to its per-VM swap file (termed *regular* swap file). Because the VM is actively writing to this file, it is difficult for another VM on another ESX server to write to this file

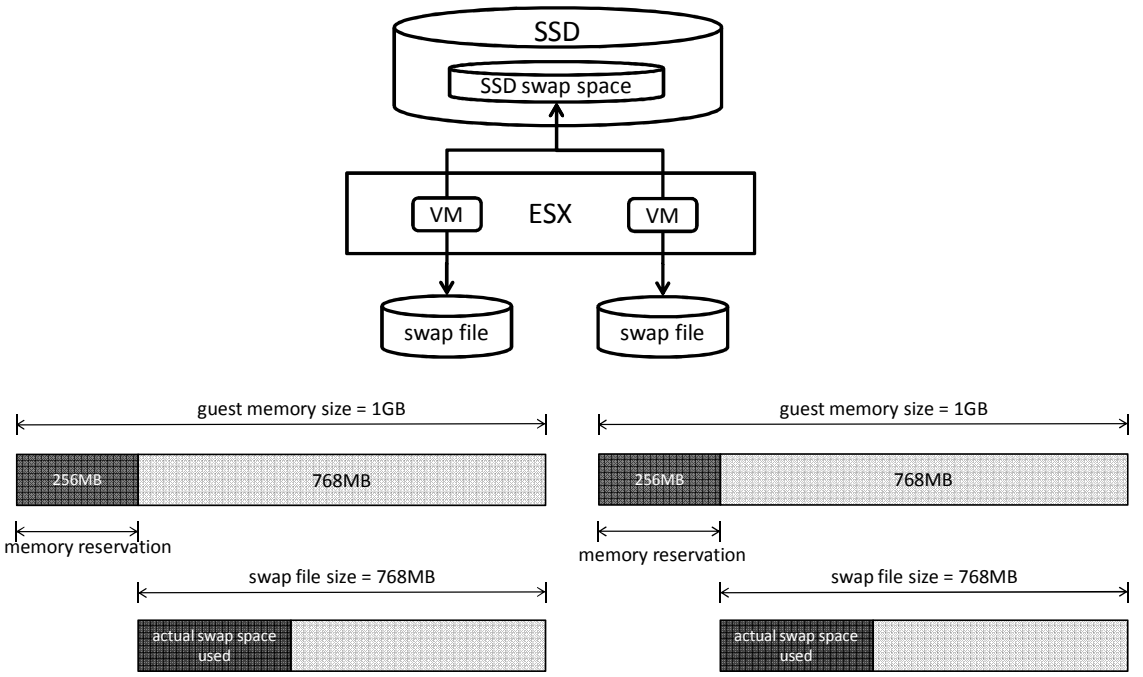


Figure 4. Schematic representation of an ESX with attached SSD and two powered-on VMs

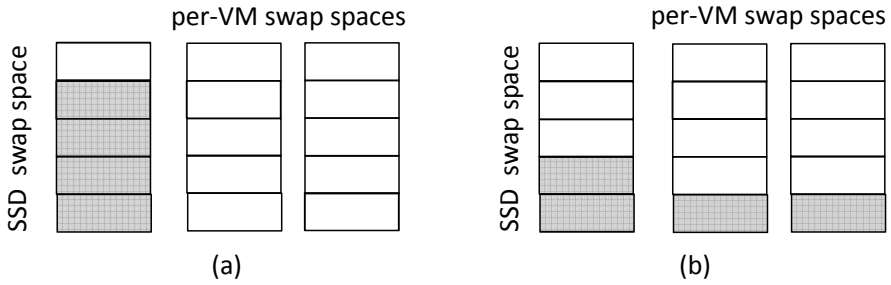


Figure 6. (a) The SSD swap space is almost full. This triggers a transfer of swapped memory pages to the per-VM swap space. (b) Pages have been transferred to the per-VM swap space such that more space is available on the SSD swap space.

without complex synchronization schemes. Hence, for simplicity ESX allows only one VM to access the per-VM swap file.

In Figure 7 (b), the source ESX server has initiated migration of the VM to the destination ESX server. The destination ESX server responds by creating a new VM with the same configuration (`.vmx`) as the source VM. This VM is not running yet. ESX has simply set up its state, in preparation of transferring the state of the source VM. The new destination VM cannot open the shared swap file on the shared storage device, because the source VM is actively using it. Instead, the destination VM creates a temporary *migration* swap file. If the destination ESX needs to reclaim memory from the destination VM by means of hypervisor-level swapping, then the swapped memory is written into the *migration* swap file.

In Figure 7 (c), the source ESX server transfers the memory, CPU and other states of the source VM to the destination ESX server. The destination ESX server sets up the state for the destination VM. During this step, the guest memory content of the source VM is transferred to the destination ESX server. This consumes memory pages on the destination. As a result, the destination ESX server can experience memory pressure and might need to reclaim memory from the destination VM. At this stage, the reclamation takes place by means of swapping only. The swapped memory is written into the *migration* swap file. During this step, the source VM is executing the guest OS. During the transfer of state from the source ESX server to the destination ESX server, guest memory that was swapped out to the swap space from the source VM is not transferred explicitly. It remains on the swap space. Subsequently, when the destination VM starts execution, it gains access to the swapped pages on this swap space.

In Figure 7 (d), the source VM's state has been completely transferred to the destination and is ready to be deleted on the source ESX server. The destination VM, after receiving complete state information from the source, has started executing the guest OS. In addition, the source VM has released access to the shared

swap file. The destination VM has opened the shared swap file as its own swap file. From Figure 7 (c), the destination VM might have swapped out guest memory pages to the migration swap file. The destination VM thus has two swap files, potentially with swapped pages in both of them. Because the regular swap file has enough space to back the unreserved memory of the VM, the temporary migration swap file is no longer required. In this step, ESX reads in all the swapped pages from the migration swap file and can swap them out to the regular swap file.

In Figure 7 (e), the contents of the migration swap file have been completely read into the VM. It is no longer required and is hence deleted. The VM has been completely migrated from the source ESX server to the destination ESX server. It is using the swap file on the shared device as its swap space.

The shared-swap vMotion is a mechanism for migrating a powered-on VM from a source ESX server to a destination ESX server when the swap space of the source and destination VM is the same and is located on a shared storage device.

4.2 Unshared-swap vMotion

Unshared-swap vMotion takes place when a powered-on VM is migrated from a source ESX server to a destination ESX server such that the per-VM swap space of the source VM is not the same as the swap space of the destination VM. It is possible, for the two swap space to reside on shared storage devices, or even on the same shared storage device. However, during and after the vMotion process the destination VM creates and uses its own swap space. The size of the swap space is given by Equation 1.

Figure 8 shows the steps during unshared-swap vMotion. In these steps, a VM is migrated from a source ESX server (left) to a destination ESX server (right). The per-VM swap file of the VM on both source and destination ESX servers are accessible to the respective VMs only.

In Figure 8 (a), which is the initial state, the source VM, on the source ESX server is linked to its per-VM swap file (termed *regular* swap file).

In Figure 8 (b), the source ESX server has initiated a migration to the destination ESX server for the VM. Similar to the shared-swap vMotion, the destination VM creates a new VM and prepares to receive the source VM's state from the source ESX server. The destination VM is linked to its own swap space. It is the permanent swap space for destination VM. It opens and has full access to this swap space in this step.

In Figure 8 (c), the source ESX server transfers the VM's state to the destination ESX server. Swapped pages from the source VM are explicitly transferred to the destination ESX server. A page that was swapped on the source VM, enters the destination VM as an in-memory page. This can subsequently be swapped to the destination swap file, if the destination ESX server decides to reclaim this page from the destination VM. During this step, the source VM is executing the guest OS.

In Figure 8 (d), the vMotion is complete. All state is transferred to the destination ESX server. The source VM is deleted. The destination VM continues to use the swap space as its permanent swap space.

Unshared-swap vMotion is a mechanism for migrating a powered-on VM from a source ESX server to a destination ESX server such that the source and destination VMs do not have the same swap file.

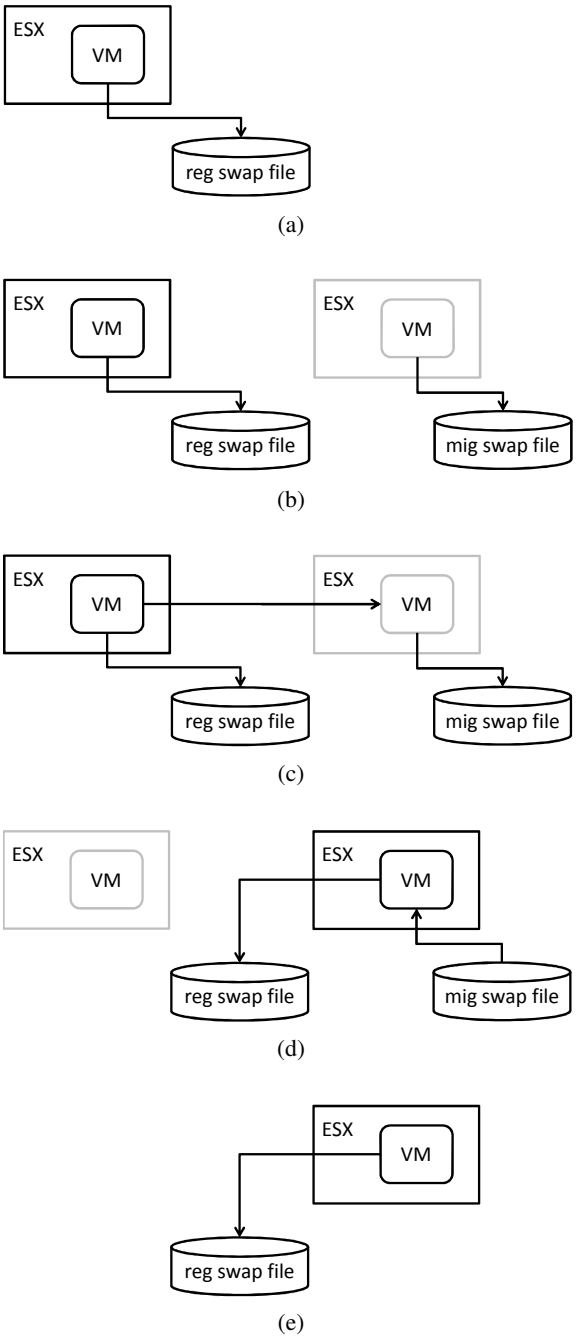


Figure 7. Schematic representation of migrating a VM from a source (left) to a destination (right) ESX server. The regular swap space is located in a swap file on a shared storage device.

5. User-world swap

Native processes in ESX are termed user-worlds (UWs). A UW is very similar to a Linux process. ESX can execute applications as a UW. Typical examples for UWs are ESX agents such as `hostd` and `vpxd`, third-party monitoring agents and the `vmx` which encapsulates the execution of a VM in ESX.

A UW can be swapped out by ESX when ESX is under memory pressure. When created, UW is tagged with one of three types – non-swappable, swappable to private swap-file or swappable to system swap file.

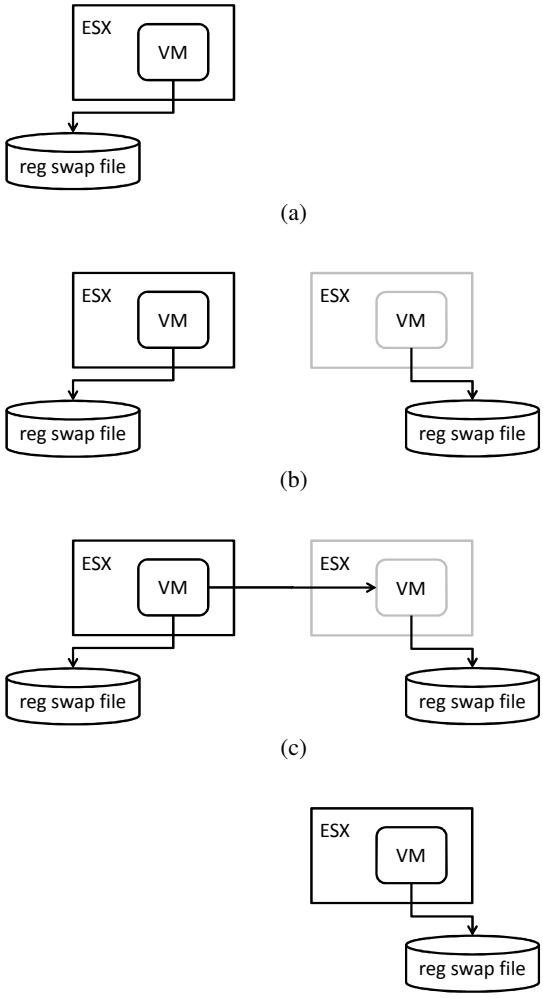


Figure 8. Schematic representation of migrating a VM from a source (left) to a destination (right) ESX server. The swap space of the source and destination VM are not on a shared storage device.

No swap: A UW started without linking to a swap space becomes non-swappable. All of its memory pages are considered pinned.
Private swap: A UW can be started with its own private swap file. In this mode, the UW is exclusively linked to a file marked as its private swap file.
System swap: ESX can be configured to create a global swap file for UWs. This swap file, termed *system swap file* can be created after ESX has booted. The location can be specified by the user, with a default size of 1GB. Only one system swap file for UWs can be created on an ESX server. When a UW is started without explicitly specifying a swap space, the UW is automatically linked to the system UW swap file.

Figure 9 shows an ESX server with UWs in different swap modes. In this example, the UW `smallapp` does not have a swap file. It is non-swappable. The UW `bigapp` has its own private swap file. The UWs `ls` are attached to the system swap file.

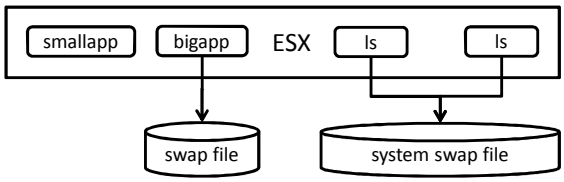


Figure 9. An ESX with UWs in different swap modes.

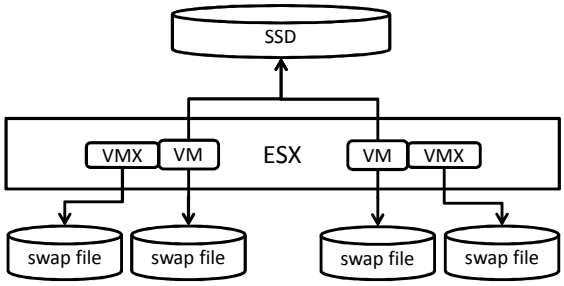


Figure 10. Schematic representation of an ESX server with two powered-on VMs. Each VM has its per-VM swap space. The `vmx` process for each VM also has its private swap file. The ESX is configured with an SSD.

6. Revisiting VM swap

For simplicity of discussion, the previous sections have ommitted aspects of the swap space layout of a VM. In this section, these aspects are discussed in greater detail.

6.1 VM swap layout

A VM is always linked to its private swap space. In addition, memory pages from a VM can be swapped out to a global swap space created on a SSD. A UW on the other hand has three modes – no swap, private swap and system swap.

A VM in ESX consists of two parts – the `vmx` process which is a UW and a virtual machine monitor and corresponding guest address space. The former, being a UW can potentially possess one of the three swap modes for a UW. The latter follows the swapping mechanism described in the preceding section for VMs.

ESX is configured to assign the private swap file mode to the `vmx` process. Hence, when a VM is powered on, two swap files are created – a private swap file for the `vmx` process and a private swap file for the VM's guest address space. Figure 10 shows the complete swap space that a VM is linked to. In Figure 10, two VMs are powered on. Each VM has its own private swap space for its `vmx` process. Each VM also has a private swap space for the guest address space. In addition, ESX is configured with a SSD. The SSD is available for storing swapped pages from both VMs.

6.2 Host-local swap space

Typically, a VM has its own private swap space. This swap space is located in the same directory as the VMs virtual disk, its configuration file and other meta-information.

ESX allows the VMs swap file to be located in a different location. ESX provides for a global setting, *host-local* swap directory, where each VM on that ESX server can store its private swap file. The host-local setting is not set by default. When it is set, all subse-

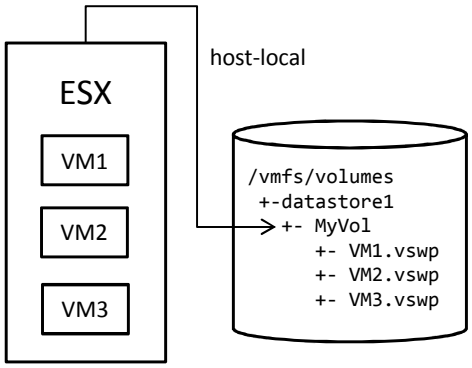


Figure 11. Schematic representation of the *host-local* swap directory setting. When set, a VM's private swap file is created in this directory. This setting does not affect powered-on VMs, SSD swap space and UW's system swap space.

quent power-on operations use the host-local swap directory to create the private swap file for the VM. Figure 11 shows a schematic representation when ESX is configured with a host-local directory. VMs powered on after this location is set have their swap space created in the host-local directory. This option does not affect the location of the SSD swap space or UW system swap space.

6.3 VM checkpoint operation

When a VM is checkpointed, using the `suspend` or `snapshot` operation, memory pages of the guest OS are written into a checkpoint file, corresponding to the `suspend` or `snapshot` operation. During this process, ESX reads the swapped memory pages from the VM's swap space before writing them to the checkpoint file. Reading the swapped pages can impact the performance of the checkpoint operation.

7. Conclusion

This article describes guest memory reclamation in ESX using hypervisor-level swapping. Swapping memory from a VM enables reliable overcommitment of ESX. ESX uses per-VM swap files for each powered-on VM. When an SSD is attached to ESX, ESX can carve out a global swap space from it to enable fast swap operations.

8. Acknowledgments

Memory overcommitment in ESX was designed and implemented by Carl Waldspurger [7]. The initial swap sub-system in ESX was implemented by Mahesh Patil. Thanks to Mukheem Ahmed for reviewing this article.

References

[1] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using KSM. In *Proceedings of the Linux Symposium*, 2009.

[2] 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. *Commun. ACM*, 53(10):85–93, Oct. 2010.

[3] M. Hines, A. Gordon, M. Silva, D. Da Silva, K. D. Ryu, and M. Ben-Yehuda. Applications Know Best: Performance-Driven Memory Overcommit with Ginkgo. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 130 –137, Dec 2011.

[4] G. Miłós, D. Murray, S. Hand, and M. Fetterman. Satori: Enlightened page sharing. In *Proceedings of the 2009 USENIX Annual Technical Conference*. USENIX Association, 2009.

[5] 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*, 2006.

[6] P. Sharma and P. Kulkarni. Singleton: system-wide page deduplication in virtual environments. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 15–26, New York, NY, 2012. ACM.

[7] C. A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, Dec. 2002.