

Memory Overcommitment in the ESX Server

Ishan Banerjee Fei Guo Kiran Tati Rajesh Venkatasubramanian

VMware Inc

{ishan, fguo, ktati, vrajesh}@vmware.com

Abstract

Virtualization of computer hardware continues to reduce the cost of operation in datacenters. It enables users to consolidate virtual hardware on a smaller number of physical hardware, thereby efficiently using hardware resources. The *consolidation ratio* is a measure of the virtual hardware that has been placed on physical hardware. A higher consolidation ratio typically indicates greater efficiency.

VMware's ESX[®] Server is a hypervisor that enables competitive memory and CPU consolidation ratios. ESX allows users to power on virtual machines (VMs) with a total configured memory that exceeds the memory available on the physical machine. This is called *memory overcommitment*. Memory overcommitment raises the consolidation ratio, increases operational efficiency and lowers total cost of operating virtual machines.

Memory overcommitment in ESX is reliable. It does not cause VMs to be suspended or terminated under any conditions. This article describes memory overcommitment in ESX, analyzes the cost of various memory reclamation techniques and empirically demonstrates that memory overcommitment induces acceptable performance penalty in workloads. Finally, best practices for implementing memory overcommitment are provided.

General Terms memory management, memory overcommitment, memory reclamation

Keywords ESX Server, memory resource management

1. Introduction

VMware's ESX Server offers competitive operational efficiency of virtual machines (VM) in the datacenter. It enables users to consolidate VMs on a physical machine while reducing cost of operation.

The *consolidation ratio* is a measure of the number of VMs placed on a physical machine. A higher consolidation ratio indicates lower cost of operation. ESX enables users to operate VMs with a high consolidation ratio. ESX Server's overcommitment technology is an enabling technology allowing users to achieve a higher consolidation ratio. Overcommitment is the ability to allocate more virtual resources than available physical resources. ESX Server offers users the ability to overcommit memory and CPU resources on a physical machine.

ESX is said to be CPU-overcommitted when the total configured virtual CPU resources of all powered-on VMs exceed the physical CPU resources on ESX. When ESX is CPU-overcommitted, it distributes physical CPU resources amongst powered-on VMs in a fair and efficient manner. Similarly, ESX is said to be memory-overcommitted when the total configured guest memory size, of all powered-on VMs, exceeds the physical memory on ESX. When

ESX is memory-overcommitted, it distributes physical memory fairly and efficiently amongst powered-on VMs. Both CPU and memory scheduling are done so as to give resources to those VMs which need them most, while reclaiming the resources from those VMs which are not actively using it.

Memory overcommitment in ESX is very similar to that in traditional operating systems (OS) such as Linux and Windows. In traditional OSES, a user may execute applications, the total mapped memory of which may exceed the amount of memory available to the OS. This is memory overcommitment. If the applications consume memory which exceeds the available physical memory, then the OS reclaims memory from some of the applications and swaps it to a swap space. It then distributes the available free memory between applications.

Similar to traditional OSES, ESX allows VMs to power on with a total configured memory size that may exceed the memory available to ESX. For the purpose of discussion in this article, the memory installed in an ESX Server is called *ESX memory*. If VMs consume all the ESX memory, then ESX will reclaim memory from VMs. It will then distribute the ESX memory, in an efficient and fair manner, to all VMs such that the memory resource is best utilized. A simple example of memory overcommitment is when two-4GB VMs are powered on in an ESX Server with 4GB of installed memory. The total configured memory of powered-on VMs is $2 * 4 = 8$ GB, while ESX memory is 4GB.

With a continuing fall in the cost of physical memory, it can be argued that ESX does not need to support memory overcommitment. However, in addition to traditional use-cases of improving the consolidation ratio, memory overcommitment can also be used in times of disaster recovery, high availability (HA) and distributed power management (DPM), to provide good performance. This technology will provide ESX with a leading edge over contemporary hypervisors.

Memory overcommitment does not necessarily lead to drastic performance loss in a guest OS or its applications. Experimental results presented in this paper with two real-life workloads show gradual performance degradation when ESX is progressively overcommitted.

This article describes memory overcommitment in ESX. It provides guidance for best-practices and talks about potential pitfalls. The remainder of this article is organized as follows. Section 2 provides background information on memory overcommitment, Section 3 describes memory overcommitment, Section 4 provides quantitative understanding of performance characteristics of memory overcommitment and Section 5 concludes the article.

2. Background and related work

Memory overcommitment enables a higher consolidation ratio in a hypervisor. Using memory overcommitment, users can consolidate VMs on a physical machine such that physical resources are utilized in an optimal manner while delivering good performance.

high	high
soft	soft
hard	hard
low	low
(a)	(b)

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

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

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

Techniques	ESX	Hyper-V	KVM	Xen
share	✓		✓	
balloon	✓	✓	✓	✓
compress	✓			
hypervisor swap	✓	✓	✓	
memory hot-add		✓		
transcendent memory*				✓

*Requires modification to guest OS kernel.

Table 3. Comparing memory overcommitment technologies in existing hypervisors. ESX implements reliable overcommitment.

For example in a virtual desktop infrastructure (VDI) deployment, a user may operate many Windows VMs, each containing a word-processing application. It is possible to overcommit a hypervisor with such VDI VMs. Since the VMs contain similar OSes and applications, many of their memory pages may contain similar content. The hypervisor will find and consolidate memory pages with identical content from these VMs, thus saving memory. This enables better utilization of memory and enables higher consolidation ratio.

Related work: Contemporary hypervisors such as Hyper-V, KVM and Xen implement different memory overcommitment, reclamation and optimization strategies. Table 3 summarizes the memory reclamation technologies implemented in existing hypervisors.

Hyper-V¹ uses dynamic memory for supporting memory overcommitment. With dynamic memory, each VM is configured with a small initial RAM when power-ed on. When the guest applications require more memory, a certain amount of memory will be hot-

added to the VM and the guest OS. When a host lacks free memory, a balloon driver, will reclaim memory from other VMs and make memory available for hot adding to the demanding VM. In the rare and restricted scenarios, Hyper-V will swap VM memory to a host swap space. Dynamic memory works for the latest versions of Windows OS. It uses only ballooning and hypervisor-level swapping to reclaim memory. ESX, on the other hand, works for all guest OSes. It also uses content-based page sharing and memory compression. This approach improves VM performance as compared to the use of only ballooning and hypervisor-level swapping.

VMs are regular processes in KVM² and therefore standard memory management techniques like swapping apply. For Linux guests, a balloon driver is installed and it is controlled by host via the balloon monitor command. Some hosts also support Kernel Sharedpage Merging (KSM) [1] which works similarly to ESX page sharing. Although KVM presents different memory reclamation techniques, it requires certain hosts and guests to support memory overcommitment. In addition, the management and policies for the interactions among the memory reclamation techniques are missing in KVM.

Xen Server³ uses a mechanism called Dynamic Memory Control (DMC) to implement memory reclamation. It works by proportionally adjusting memory among running VMs based on pre-defined minimum and maximum memory. VMs generally run with maximum memory and the memory can be reclaimed via a balloon driver when memory contention in the host occurs. However, Xen does not provide a way to overcommit the host physical memory, hence its consolidation ratio is largely limited. Unlike other hypervisors, Xen provides a transcendent memory management mechanism to manage all host idle memory and guest idle memory. The idle memory is collected into a pool and distributed based on the demand of running VMs. This approach requires the guest OS to be paravirtualized, and only works well for guests with non-concurrent memory pressure.

When compared to existing hypervisors, ESX allows for reliable memory overcommitment to achieve high consolidation ratio with no requirements/modifications for running guests. It implements various memory reclamation techniques to enable overcommitment and manages them in an efficient manner to mitigate possible performance penalties to VMs.

Work on optimal use of host memory in a hypervisor has also been demonstrated by the research community. An optimization for the KSM technique has been attempted in KVM with Singleton [6]. Sub-page level page sharing using patching has been demonstrated in Xen with Difference Engine [2]. Paravirtualized guest and sharing of pages read from storage devices have been shown using Xen in Satori [4].

Memory ballooning has been demonstrated for the z/VM hypervisor using CMM [5]. Ginkgo [3] implements a hypervisor-independent overcommitment framework allowing Java applications to optimize their memory footprint. All these works target specific aspects of memory overcommitment challenges in virtualization environment. They are valuable references for future optimizations in ESX memory management.

Background: Memory overcommitment in ESX is reliable (see Table 3). This implies that VMs will not be prematurely terminated or suspended owing to memory overcommitment. Memory overcommitment in ESX is theoretically limited by the overhead memory of ESX. ESX guarantees reliability of operation under all levels of overcommitment.

¹ www.microsoft.com/hyper-v-server/

² www.xen.org/

³ www.linux-kvm.org/

When ESX is memory overcommitted, it allocates memory to those powered-on VMs that need it most and will perform better with more memory. At the same time, ESX reclaims memory from those VMs which are not actively using it. Memory reclamation is therefore an integral component of memory overcommitment. ESX uses different memory reclamation techniques to reclaim memory from VMs. The memory reclamation techniques are – transparent page sharing, memory ballooning, memory compression and memory swapping.

ESX has an associated memory *state* which is determined by the amount of free ESX memory at a given time. The states are – *high*, *soft*, *hard* and *low*. Table 1 shows the state threshold of ESX. Each threshold is internally split into two sub-thresholds to avoid oscillation of ESX memory state near the threshold. At each memory state ESX utilizes a combination of memory reclamation techniques to reclaim memory. This is shown in Table 2.

Transparent page sharing (page sharing) is a passive and opportunistic memory reclamation technique. It operates on a powered-on VM at all memory states, throughout its lifetime, looking for opportunities to collapse different memory pages with identical content into one memory page. This method greatly reduces the memory footprint of VMs with common memory content. For example, if an ESX has many VMs executing word processing applications, then ESX may transparently apply page sharing to those VMs and collapse the text and data content of these applications, thereby reducing the footprint of all those VMs. The collapsed memory is freed by ESX and made available for powering-on more VMs. This raises the consolidation ratio of ESX and enables higher overcommitment. In addition, if the shared pages are not subsequently written into, by the VMs, then they remain shared for a prolonged time, maintaining the reduced footprint of the VM.

Memory ballooning is an active method for reclaiming idle memory from VMs. It is used when ESX is in the *soft* state. If a VM has consumed memory pages, but is not subsequently using them in an active manner, ESX attempts to reclaim them from the VM using ballooning. In this method, an OS-specific *balloon driver* inside the VM allocates memory from the OS kernel. It then hands the memory to ESX, which is then free to re-allocate it to another VM which might be actively requiring memory. The balloon driver effectively utilizes the memory management policy of the guest OS to reclaim idle memory pages. The guest OS typically reclaims idle memory inside the guest OS and if required swaps them to its own swap space.

When ESX enters the *hard* state, it actively and aggressively reclaims memory from VMs by swapping out memory to a swap space. During this step, if ESX determines that a memory page is sharable or compressible, then that page is shared or compressed instead. The reclamation done by ESX using swapping is different from that done by the guest OS inside the VM. The guest OS may swap out guest memory pages to its own swap space – for example `/swap` or `pagefile.sys`. ESX uses hypervisor-level swapping to reclaim memory from a VM into its own swap space.

The *low* state is similar to the *hard* state. In addition, to compressing and swapping memory pages, ESX may block certain VMs from allocating memory in this state. It aggressively reclaims memory from VMs, until ESX moves into the *hard* state.

Page sharing is a passive memory reclamation technique that operates continuously on a powered-on VM. The remaining techniques are active ones that operate when free memory in ESX is low. Also, page sharing, ballooning and compression are opportunistic techniques. They do not guarantee memory reclamation from VMs. For example, a VM may not have sharable content, the balloon driver may not be installed or its memory pages may not yield good compression. Reclamation by swapping is a guaranteed method for reclaiming memory from VMs.

In summary, ESX allows for reliable memory overcommitment to achieve a higher consolidation ratio. It implements various memory reclamation techniques to enable overcommitment while improving efficiency and lowering cost of operation of VMs. The next section describes memory overcommitment.

3. Memory overcommitment

Memory overcommitment enables a higher consolidation ratio of VMs on an ESX Server. It reduces cost of operation while utilizing compute resources efficiently. This section describes memory overcommitment and its performance characteristics.

3.1 Definitions

ESX is said to be memory overcommitted when VMs are powered on such that their total configured memory size is greater than ESX memory. Figure 1 shows an example of memory overcommitment on ESX.

Figure 1 (a) shows the schematic diagram of a memory-undercommitted ESX Server. In this example, ESX memory is 4GB. Two VMs, each with a configured memory size of 1GB are powered on. The total configured memory size of powered-on VMs is therefore 2GB, which is less than 4GB. Hence ESX is considered to be memory undercommitted.

Figure 1 (b) shows the schematic diagram of a memory-overcommitted ESX Server. In this example, ESX memory is 4GB. Three VMs, with configured memory sizes of 1GB, 2.5GB and 1GB are powered on. The total configured memory size of powered-on VMs is therefore 4.5GB which is more than 4GB. Hence ESX is considered to be memory overcommitted.

The scenarios described above omit the memory overhead consumed by ESX. ESX consumes a fixed amount of memory for its own text and data structures. In addition, it consumes overhead memory for each powered-on VM.

ESX also reserves a small amount of memory called *minfree*. This amount is a buffer against rapid allocations by memory consumers. ESX is in *high* state as long as there is at least this amount of memory free. If the free memory dips below this value then ESX is no longer in *high* state and it begins to actively reclaim memory.

Figure 1 (c) shows the schematic diagram representing an undercommitted ESX when overhead memory is taken into account. In this diagram, the overhead memory consumed by ESX for itself and for each powered-on VM is shown. Figure 1 (d) shows the schematic diagram representing an overcommitted ESX when overhead memory is taken into account.

Figure 1 (e) shows the theoretical limit of memory overcommitment in ESX. In this case, all of ESX memory is consumed by ESX overhead and per-VM overhead. VMs will be able to power-on and boot, however, execution of the VM will be extremely slow.

For simplicity of discussion and calculation, the definition of overcommitment from Figure 1 (b) is followed. Overhead memory is ignored for defining memory overcommitment. From this figure:

$$overcommit = \sum_{v \in V} memsize / ESXmemory \quad (1)$$

where:

$overcommit$	memory overcommitment factor
V	powered-on VMs in ESX
$memsize$	configured memory size of v
$ESXmemory$	total installed ESX memory

The representation from this figure is used in the remainder of this article.

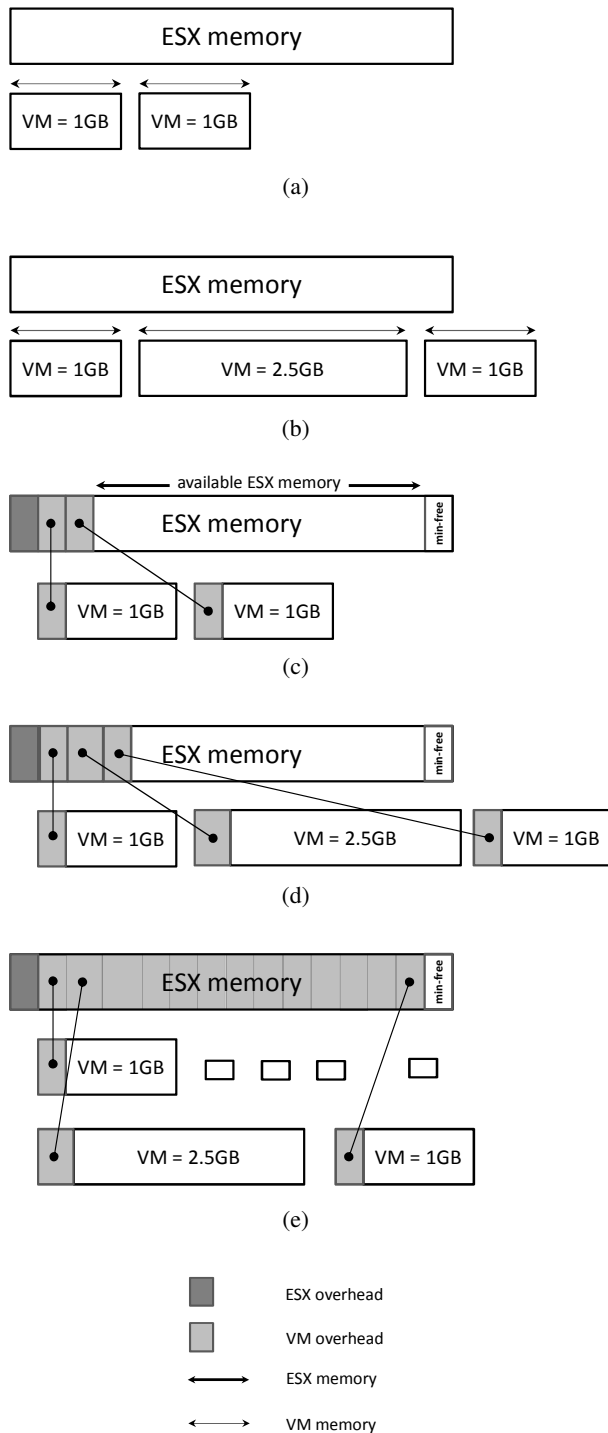


Figure 1. Memory overcommitment shown with and without overhead memory. (a) Undercommitted ESX (b) Overcommitted ESX. This model is typically followed when describing overcommitment (c) Undercommitted ESX shown with overhead memory (d) Overcommitted ESX shown with overhead memory. (e) Limit of memory overcommitment.

To understand memory overcommitment and its effect on VM and applications, *mapped*, *consumed* and *working set* memory are described.

3.2 Mapped memory

The definition of memory overcommitment does not consider the memory consumption or memory access characteristic of the powered-on VMs. Immediately after a VM is powered on, it does not have any memory pages allocated to it. Subsequently, as the guest OS boots, the VM accesses pages in its memory address space by reading or writing into it. ESX allocates physical memory pages to back the virtual address space of the VM during this access. Gradually, as the guest OS completes booting and applications are launched inside the VM, more pages in the virtual address space are backed by physical memory pages. During the lifetime of the VM, the VM may or may not access all pages in its virtual address space.

Windows, for example, writes the *zero*⁴ pattern to the complete address space of the VM. This causes ESX to allocate memory pages for the complete address space by the time Windows has completed booting. On the other hand, Linux does not access the complete address space of the VM when it boots. It accesses only memory pages required to load the OS.

All memory pages of a VM which are ever accessed by a VM are considered *mapped* by ESX. A mapped memory page is backed by a physical memory page by ESX during the very first access by the VM. A mapped page may subsequently be reclaimed by ESX. It is considered as mapped through the lifetime of the VM.

When ESX is overcommitted, the total mapped pages by all VMs may or may not exceed ESX memory. Hence, it is possible for ESX to be memory overcommitted, but at the same time, owing to the nature of the guest OS and its applications, the total mapped memory remains within ESX memory. In such a scenario, ESX does not actively reclaim memory from VMs, and VM performance is not affected by memory reclamation.

Mapped memory is illustrated in Figure 2. This figure uses the representation from Figure 1 (b) where ESX is memory overcommitted.

In Figure 2 (a) the memory mapped by each VM is shown. The total mapped memory of all VMs is less than ESX memory. In this case, ESX is overcommitted. However, the total mapped memory is less than ESX memory. ESX will not actively reclaim memory in this case.

In Figure 2 (b) the memory mapped by each VM is shown. In this case, ESX is overcommitted. In addition, the total mapped memory in ESX exceeds ESX memory. ESX may or may not actively reclaim memory from the VMs. This depends on the current *consumed memory* of each VM.

3.3 Consumed memory

A VM is considered to be *consuming* a physical memory page when a physical memory page is used to back a memory page in its address space. A memory page of the VM may exist in different states with ESX. They are as follows:

Regular A regular memory page in the address space of a VM is one which is backed by one physical page in ESX.

Shared A memory page marked as shared in a VM may be shared with many other memory pages. If a memory page is being shared with n other memory pages, then each memory page is considered to be consuming $1/n$ of a whole physical page.

Compressed VM pages which are compressed typically consume $1/2$ or $1/4$ of a physical memory page.

⁴ A memory page with all 0x00 content is a *zero* page.

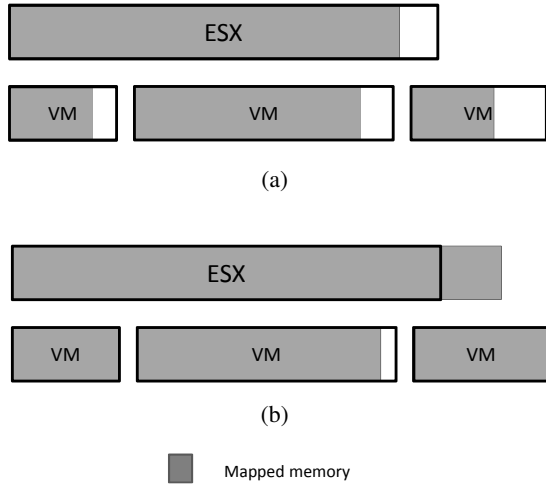


Figure 2. Mapped memory for an overcommitted ESX. Mapped region is shaded. The mapped region in ESX is the sum of mapped region in the VMs. (a) Total mapped memory is less than ESX memory (b) Total mapped memory is more than ESX memory.

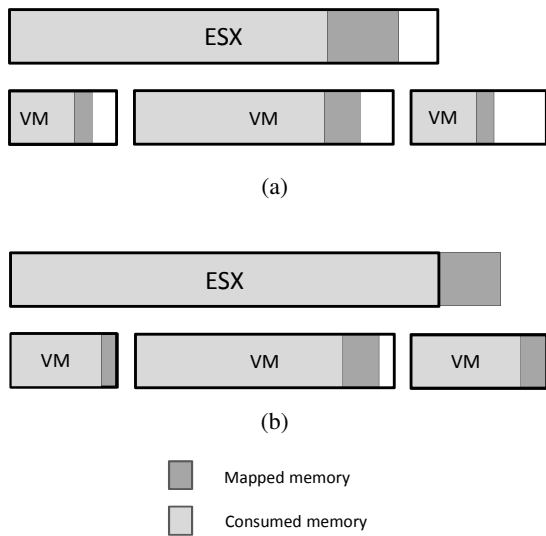


Figure 3. Consumed memory for an overcommitted ESX. Consumed and mapped regions are shaded. The consumed (and mapped) region in ESX is the sum of consumed (mapped) region in the VMs. (a) Total consumed and mapped memory is less than ESX memory (b) Total mapped memory is more than ESX memory, total consumed memory is equal to ESX memory.

Ballooned Ballooned memory pages are not backed by any physical page.
Swapped Swapped memory pages are not backed by any physical page.

Consumed memory is illustrated in Figure 3. This figure uses the representation from Figure 1 (b) where ESX is memory overcommitted.

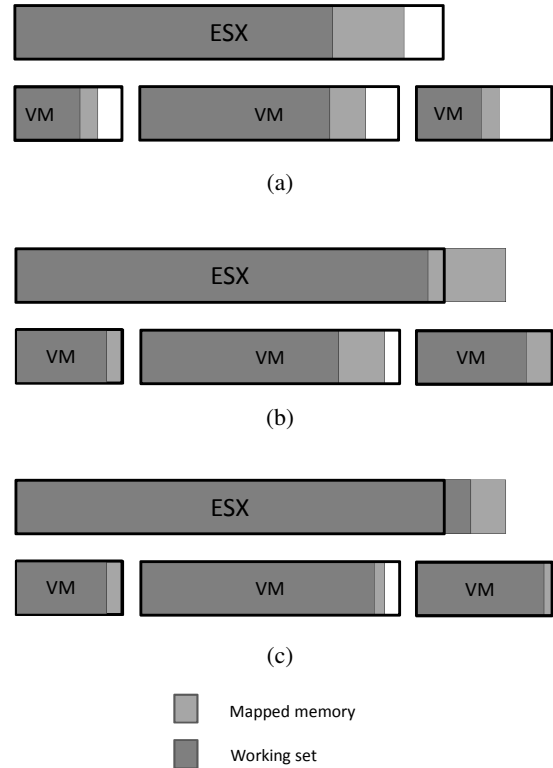


Figure 4. Working set of a VM in memory overcommitted ESX. Consumed memory is not shown. (a) Mapped and working set is less than ESX memory. (b) Mapped memory is more than ESX memory, however, the working set is less than ESX memory. (c) Mapped and working set is greater than ESX memory.

In Figure 3 (a) the mapped and consumed memory of powered-on VMs and ESX is shown. ESX is overcommitted. However, the total mapped memory is less than ESX memory. In addition, the total consumed memory is less than the total mapped memory. The consumption is lower than mapped since some memory pages may have been shared, ballooned, compressed or swapped. In this state, ESX will not actively reclaim memory from VMs. However, the VMs may possess ballooned, compressed or swapped memory. This is because ESX may have reclaimed memory earlier from these VMs owing to the memory state at that time.

The total consumption of all VMs taken together cannot exceed ESX memory. This is shown in Figure 3 (b). In this figure, ESX is overcommitted. In addition, the total mapped memory is greater than ESX memory. However, whenever, VMs attempt to consume more memory than ESX memory, ESX will reclaim memory from the VMs and redistribute the ESX memory amongst all VMs. This prevents ESX from running out of memory. In this state, ESX is likely going to actively reclaim memory from VMs to prevent memory exhaustion.

3.4 Working set

The *working set* of a VM is the set of memory pages which are being “actively” accessed by the VM. The size of the working set depends on the precise definition of “active”. A precise definition is omitted in this article since it requires analysis and discussion beyond the scope of this article. The working set of a VM depends on the application it is executing and its memory access character-

istics. The working set may change slowly in size and location for some applications while for other applications, the change may be rapid.

Memory pages in a working set are either shared pages or regular pages. If the working set accesses a compressed, ballooned or swapped page or writes into a shared page, then that page will be *page-faulted* and immediately backed by a physical page by ESX. This page-fault incurs a temporal cost. The temporal cost is typically high for pages swapped onto a spinning disk. This high cost of page-fault may cause the VM to be temporarily stalled. This in turn adversely affects the performance of the VM and its applications.

Working set memory is illustrated in Figure 4. This figure uses the representation from Figure 1 (b) where ESX is memory overcommitted.

In Figure 4 (a) the mapped memory and working set of powered-on VMs are shown. In this figure, both values are less than the ESX memory. Consumed memory (not shown) will be less than or equal to mapped memory, hence less than ESX memory. In this case, there will not be active memory reclamation by ESX from any of the VMs.

Note, that the working set is not dependent on the consumed memory. The working set contains memory pages which are being read or written into. The consumed memory may be less than the working set if many pages are shared. Shared pages may be read without invoking a copy-on-write of the shared page but will count towards the working set. On the other hand, consumed memory may be more than the working set if reclamation did not take place and the VM is accessing few memory pages. Hence, consumed memory and working set memory do not have a dependency.

In Figure 4 (b) the mapped memory is more than ESX memory. The consumed memory is not shown. It is less than or equal to ESX memory, but may be different from the working set. However, the working set is less than ESX memory. Hence, in steady state, when the working set is not changing, all pages of the working set will be backed by a physical page.

In Figure 4 (c), the mapped memory and working set are greater than ESX memory. The consumed memory will not exceed the ESX memory. ESX will reclaim memory from VMs, such that memory does not get completely exhausted, and distribute the memory amongst VMs to service page-faults. This case represents a case of memory thrashing. Since the working set of all VMs is more than ESX memory, the VMs will continue to access pages in its working set causing them to be page-faulted. At the same time ESX will continue to reclaim memory from VMs. If the access pattern from VMs is a read-operation of a shared page, then there will be no page-fault.

Figure 4 (c) represents a case where performance of the VMs may suffer owing to thrashing. Since the temporal wait cost of page-fault is highest for spinning disk, pages swapped to a spinning disk will trigger the largest performance loss. Write access to shared pages, access to compressed and ballooned pages and access to pages swapped to SSD⁵ will incur significantly lower temporal wait cost.

3.5 What is memory overcommitment ?

From Sections 3.1– 3.4, it can be seen that an ESX can be memory overcommitted while the total mapped, consumed and working set remain within ESX memory. Memory overcommitment, as defined, does not imply that the performance of a VM and its applications will be affected.

Performance of an application in a VM largely depends on its working set. This is because, in steady state, the working set of a

VM is backed by physical memory. Access to a working set backed by physical memory is fast and does not incur temporal wait cost for page-faulting reclaimed pages. Hence, if the total working set of VMs remains within ESX memory, then the VMs do not experience significant performance loss.

Typically, a working state is not static; it changes in size and locality. When the working set changes, the VM may need to page-fault a reclaimed page. This incurs a temporal wait cost, leading to a performance loss. However, for VMs with stable working sets, the frequency and duration of changes in working set are low. Hence the VM may experience a reduced performance for only brief durations when the working set changes.

3.6 Cost of memory overcommitment

Memory overcommitment incurs certain cost in terms of compute resource as well as VM performance. This section provides a qualitative understanding of the different sources of cost and their magnitude.

When ESX is memory overcommitted and powered-on VMs attempt to consume more memory than ESX memory, then ESX will begin to actively reclaim memory from VMs. Hence memory reclamation is an integral component of memory overcommitment.

Table 4 shows the memory reclamation techniques and their associated cost. Each technique incurs a cost of reclamation – when a page is reclaimed – and a cost of page-fault – when a VM accesses a reclaimed page. The number of * symbols qualitatively indicate the magnitude of the cost.

A **Memory** cost indicates that the technique consumes memory meta-data overhead. A **CPU** cost indicates that the technique consumes non-trivial CPU resources. A **Storage** cost indicates that the technique consumes storage space or bandwidth. A **Wait** cost indicates that the VM incurs a hypervisor-level page-fault cost as a result of the technique. This may lead to the guest application stalling and hence lead to a drop in its performance. The reclamation and page-fault costs are described as follows:

Page sharing :

Reclamation cost: Page sharing is a continuous process which opportunistically shares VM memory pages.

Memory This technique incurs an ESX-wide memory cost for storing page-sharing meta-data which is a part of ESX overhead memory.

CPU Page sharing incurs a nominal CPU cost on a per-VM basis. This cost is typically very small and does not impact the performance of VM applications or benchmarks.

Page-fault cost: When a shared page is read by a VM, it is accessed by the VM in a read-only manner. Hence that shared pages does not need to be page-faulted. Hence there is no page-fault cost. A write access to a shared page incurs a cost.

CPU When a shared page is written to, by a VM, ESX must allocate a new page and replicate the shared content before allowing the write access from the VM. This allocation incurs a CPU cost. Typically, this cost is very low and does not significantly affect VM applications and benchmarks.

Wait The copy-on-write operation when a VM accesses a shared page with write access is fairly fast. VM applications accessing the page do not incur noticeable temporal cost.

Ballooning :

Reclamation cost: Memory is ballooned from a VM using a balloon driver residing inside the guest OS. When the balloon driver expands, it may induce the guest OS to reclaim memory from guest applications.

⁵ Solid state device

CPU Ballooning incurs a CPU cost on a per-VM basis since it induces memory allocation and reclamation inside the VM.
Storage The guest OS may swap out memory pages to the guest swap space. This incurs storage space and storage bandwidth cost.

Page-fault cost:

CPU A ballooned page acquired by the balloon driver may subsequently be released by it. The guest OS or application may then allocate and access it. This incurs a page-fault in the guest OS as well as ESX. The page-fault incurs a low CPU cost since a memory page simply needs to be allocated.

Storage During reclamation by ballooning, application pages may have been swapped out by the guest OS. When the application attempts to access that page, the guest OS needs to swap it in. This incurs a storage bandwidth cost.

Wait A temporal wait cost may be incurred by an application if its pages were swapped out by the guest OS. The wait cost of swapping in a memory page by the guest OS incurs a smaller overall wait cost to the application than a hypervisor-level swap-in. This is because during a page fault in the guest OS, by one thread, the guest OS may schedule another thread. However, if ESX is swapping in a page, then it may deschedule the entire VM. This is because ESX cannot reschedule guest OS threads.

Compression :

Reclamation cost: Memory is compressed in a VM by compressing a full guest memory page such that it consumes 1/2 or 1/4 physical memory page. There is effectively no memory cost, since every successful compression releases memory.

CPU A CPU cost is incurred for every attempted compression. The CPU cost is typically low and is charged to the VM whose memory is being compressed. The CPU cost is however, more than that for page sharing. It may lead to noticeably reduced VM performance and may affect benchmarks.

Page-fault cost: When a compressed memory page is accessed by the VM, ESX must allocate a new page and de-compress the page before allowing access to the VM.

CPU De-compression incurs a CPU cost.

Wait The VM also waits until the page is de-compressed. This is typically not very high since the de-compression takes place in-memory.

Swap :

Reclamation cost: ESX swaps out memory pages from a VM to avoid memory exhaustion. The swap-out process takes place asynchronously to the execution of the VM and its application. Hence, the VM and its applications do not incur a temporal wait cost.

Storage Swapping out memory pages from a VM incurs storage space and storage bandwidth cost.

Page-fault cost: When a VM page-faults a swapped page, ESX must read the page from the swap space synchronously before allowing the VM to access the page.

Storage This incurs a storage bandwidth cost.

Wait The VM also incurs a temporal wait cost while the swapped page is synchronously read from storage. This wait cost is highest for a swap space located on spinning disk. Swap space located on SSD incurs lower wait cost.

	share	balloon	compress	swap (SSD)	swap (disk)
<hr/>					
Reclaim					
Memory	*				
CPU	*	*	**		
Storage		* ¹		*	*
Wait					
<hr/>					
Page-fault					
Memory					
CPU	*	*	**		
Storage		* ¹		*	*
Wait	*	*** ¹	*	**	*****

1 - indicates that this cost *may* be incurred under certain conditions.

Table 4. Cost of reclaiming memory from VMs and cost of page-fault when VM accesses a reclaimed page. Number of * qualitatively indicates the magnitude of cost. They are not mathematically proportional.

The cost of reclamation and page-fault vary significantly among different reclamation techniques. Reclamation by ballooning may incur a storage cost only in certain cases. Reclamation by swapping always incurs storage cost. Similarly, page-fault owing to reclamation incurs a temporal wait cost in all cases. Page-fault of a swapped page incurs the highest temporal wait cost. Table 4 shows the cost of reclamation as well as the cost of page-fault incurred by a VM on a reclaimed memory page.

The reclamation itself does not impact VM and application performance significantly, since the temporal wait cost of reclamation is zero for all techniques. Some techniques have a low CPU cost. This cost may be charged to the VM leading to slightly reduced performance⁶.

However, during page-fault temporal wait cost exists for all techniques. This affects VM and application performance. Write access to shared pages and access to compressed, ballooned pages incur the least performance cost. Access to swapped pages, especially pages swapped to spinning disk, incurs the highest performance cost.

Section 3 described memory overcommitment in ESX. It defined various terms – *mapped*, *consumed* and *working set* memory – and showed their relation to memory overcommitment. It also showed that memory overcommitment does not necessarily impact VM and application performance. This section also provided a qualitative analysis of how memory overcommitment may impact VM and application performance depending on the VM’s memory content and access characteristics.

The next section provides a quantitative description of overcommitment.

4. Performance

This section provides a quantitative demonstration of memory overcommitment in ESX. Simple applications are shown to work using memory overcommitment. This section does not provide a comparative analysis using software benchmarks.

⁶ CPU cost of reclamation may affect performance only if ESX is operating at 100% CPU load

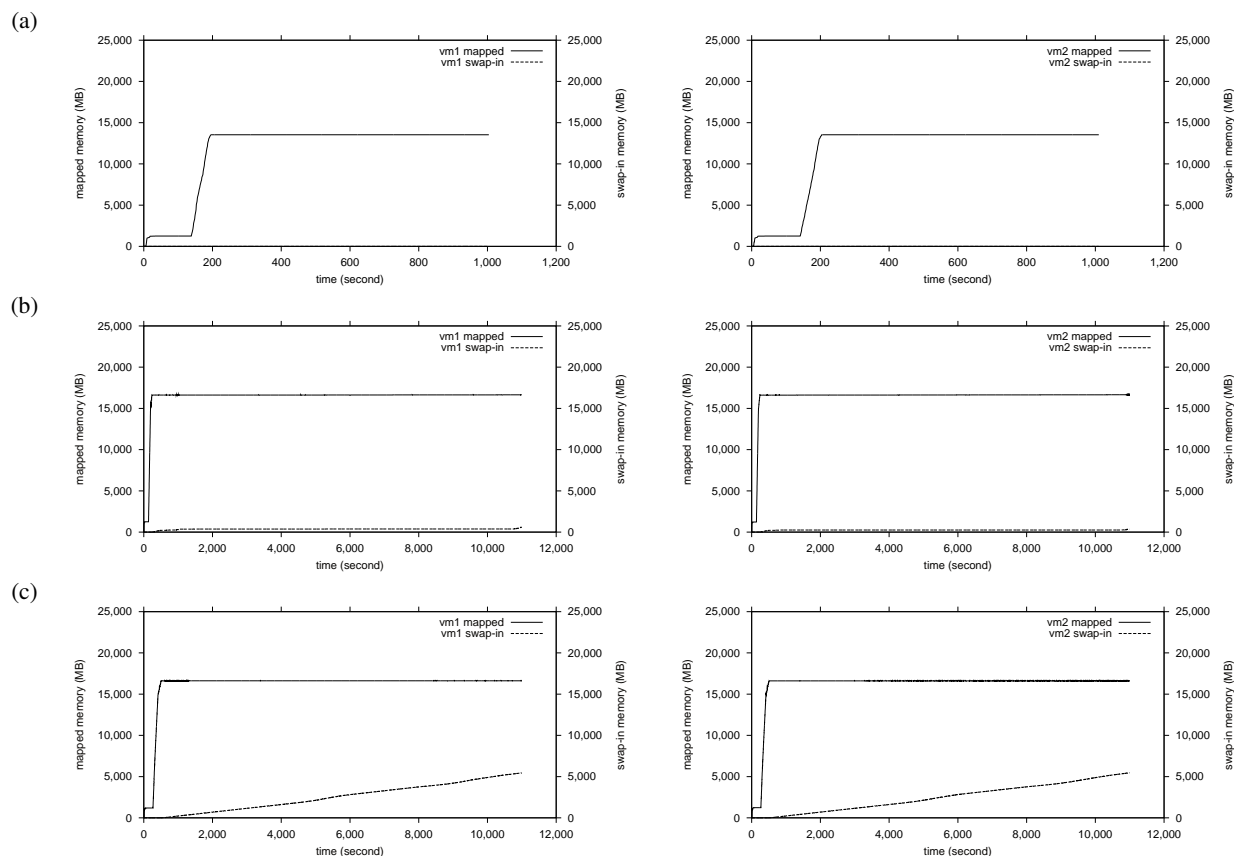


Figure 5. Effect of working set on memory reclamation and page-fault. Available ESX memory=28GB. (a) Mapped, working set < available ESX memory. (b) Mapped > available ESX memory, working set < available ESX memory. (c) Mapped, working set > available ESX memory.

4.1 Microbenchmark

This set of experiments was conducted on a development build of ESX 6.0. The ESX Server has 32GB RAM and 4 quad-core AMD CPUs configured in a 4-NUMA node configuration. The VMs used in the experiments are 2-vCPU, 24GB and contain RHEL6 OS. The ESX Server consumes about 4 GB memory – ESX memory overhead, per-VM memory overhead and reserved *minfree*. This leaves about 28GB for allocating to VMs. For these experiments, ESX memory is 32GB. However, ESX will actively reclaim memory from VMs when VMs consume more than 28GB (available ESX memory). RHEL6 consumes about 1GB memory when booted and idle. This usage will contribute to the mapped and consumed memory of the VM.

These experiments demonstrate performance of VMs when the total working set varies compared to available ESX memory. Figure 5 shows the results from three experiments. For the purpose of demonstration and simplicity the three reclamation techniques – page sharing, ballooning and compression are disabled. Hypervisor-level memory swapping is the only active memory reclamation technique in this experiment. This reclamation will start when VMs consumed more than available ESX memory.

In these experiments, presence of hypervisor-level page-fault is an indicator of performance loss. Figures show cumulative swap-in (page-fault) values. When this value rises, VM will experience temporal wait cost. When it does not rise, there is no temporal wait

cost and hence no performance loss.

Experiment a: In Figure 5 (a) an experiment is conducted similar to Figure 4 (a). In this experiment, the total mapped and total working set memory always remain less than available ESX memory.

Two VMs, each with a configured memory size of 24GB, are powered-on. Since ESX memory is 32GB, the overcommitment factor is $(2 * 24) / 32 = 1.5$. Each VM executes an identical hand-crafted workload that is a memory stress program. This program allocates 12GB memory, writes random pattern into all of this allocated memory and continuously reads this memory in a round-robin manner. This workload is executed in both VMs. Hence a total of 24GB memory is mapped and actively used by the VMs. The workload is executed for 900 seconds.

The figure shows *mapped* memory for each VM. It also shows the cumulative *swap-in* for each VM. The X-axis in this figure shows time in seconds, Y-axis (left) shows mapped memory in MB and Y-axis (right) shows cumulative swap-in. The memory mapped by each VM is about 13GB (workload=12GB, RHEL6=1GB), the total being 26GB. This is less than available ESX memory. It can be seen from this figure that there is no swap-in (page-fault) activity at any time. Hence there is no performance loss. The memory mapped by the VMs rise as the workload allocates memory. Subsequently, as the workload accesses the memory in a round-robin manner, all of the memory is backed by physical memory pages by ESX.

This experiment demonstrates that although memory is over-committed, VMs mapping and actively using less than available ESX memory will not be subjected to active memory reclamation and hence will not experience any performance loss.

Experiment b: In Figure 5 (b) an experiment is conducted similar to Figure 4 (b). In this experiment, the total mapped memory exceeds available ESX memory, while the total working set memory is less than available ESX memory.

Two VMs, each with a configured memory size of 24GB, are powered-on. The resulting memory overcommitment is 1.5. The workload in each VM allocates 15GB of memory and writes a random pattern into it. It then reads a fixed block, 12.5% of 15GB (=1.875GB) in size, from this 15GB in a round-robin manner for 2 iterations. Thereafter, the same fixed block is read in a round-robin manner for 10,000 seconds. This workload is executed in both VMs. The memory mapped by each VM is about 16GB (workload=15GB, RHEL=1GB), the total being 32GB. This exceeds available ESX memory. The working set is thereafter a total of $2 * 1.875 = 3.75$ GB.

The figure shows *mapped* memory for each VM and cumulative *swap-in*. The X-axis in this figure shows time in seconds, Y-axis (left) shows mapped memory in MB and Y-axis (right) shows cumulative swap-in. It can be seen from this figure that as the workload maps memory, the mapped memory rises. Thereafter, there is an initial rising swap-in activity as the working set is page-faulted. After the working set is page-faulted, the cumulative swap-in is steady.

This experiment demonstrates that although memory is over-committed and VMs have mapped more memory than available ESX memory, VMs will perform better when their working set is smaller than available ESX memory.

Experiment c: In Figure 5 (c) an experiment is conducted similar to Figure 4 (c). In this experiment, the total mapped memory as well as the total working set exceeds available ESX memory.

Two VMs, each with a configured memory size of 24GB, are powered on. The overcommitment factor is 1.5. Each VM runs a workload that allocates 15GB of memory and writes a random pattern into it. It then reads all 15GB memory continuously in a round robin manner for 2 iterations. The memory mapped by each VM is about 16GB (workload=15GB, RHEL6=1GB), the total being 32GB. The working set is also 32GB. Both exceed available ESX memory.

The figure shows *mapped* memory for each VM and cumulative *swap-in*. The X-axis in this figure shows time in seconds, Y-axis (left) shows mapped memory in MB and Y-axis (right) shows cumulative swap-in. It can be seen from this figure that a steady page-fault is maintained once the workloads have mapped the target memory. This indicates that as the workloads are accessing memory, they are experiencing page-faults. ESX is continuously reclaiming memory from each VM as the VM page-faults its working set.

This experiment demonstrates that workloads will experience steady page-faults and hence performance loss when the working set exceeds available ESX memory. Note that if the working set read-accesses shared pages and page sharing is enabled (default ESX behavior) then a page-fault is avoided.

In this section experiments were designed to highlight the basic working of overcommitment. In these experiments, page sharing, ballooning and compression were disabled for simplicity. These reclamation techniques reclaim memory effectively before reclamation by swapping can take place. Since page faults to shared, ballooned and compressed memory have a lower temporal wait cost,

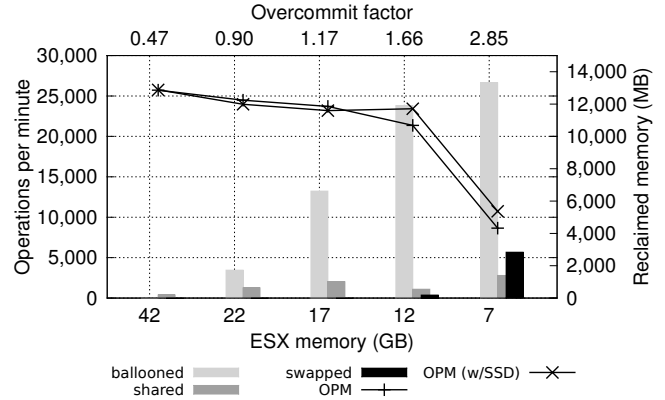


Figure 6. DVDStore2. Operations per minute (OPM) of five DVDstore2 VMs when changing ESX memory from 42GB to 7GB.

the workload performance is better. This will be demonstrated in the next section.

4.2 Real workloads

In this section, the experiments are conducted using vSphere 6.0. The ESX Server has 72GB RAM and 4 quad-core AMD CPUs. The workloads used to evaluate memory overcommitment performance are DVDstore2⁷ and a VDI workload. Experiments were conducted with default memory management configurations for pshare, balloon and compression.

Experiment d: The DVDstore2 workload simulates online database operations for ordering DVDs. In this experiment, 5 DVDstore2 VMs are used, each configured with 4 vCPUs and 4GB memory. Each VM contains Windows Server 2008 OS and SQL Server 2008. The performance metric is the total operations per minute of all 5 VMs.

Since five 4GB VMs are used, the total memory size of all powered-on VMs is 20GB. However, the ESX Server contains 72GB installed RAM and was effectively undercommitted. Hence, memory overcommitment was simulated with the use of a memory hog VM. The memory hog VM had full memory reservation. Its configured memory size was progressively increased in each run. This effectively reduced the ESX memory available to the DVD-Store2 VMs.

Figure 6 shows the performance of DVDstore2 VMs for ESX memory values of {42, 22, 17, 12, 7} GB. At these data points, the memory hog VM has a configured memory size of {30, 50, 55, 60, 65} GB. Owing to the reduced simulated ESX memory, the simulated overcommitment factor, using Equation 1, (w/ total VM memory size of 20GB) was {0.47, 0.90, 1.17, 1.66, 2.85}.

The Y-axis (right) shows the total amount of ballooned and swapped memory, in MB, from all VMs. It can be seen that the amount of ballooned memory increases gradually without significant hypervisor-level swapping as ESX memory is reduced from 42GB to 12GB. The Y-axis (left) shows the operations per minute (OPM) executed by the workload. The rising ballooned memory contributes to reduced application performance as ESX memory is reduced from 42GB to 12GB. This is owing to the wait cost to the application as some of its memory is swapped out by the guest OS. Between the X-axis points 42 and 12, the OPM decreases from 25,719 to 21,365 (about 17%) owing to ballooning. The overcommitment factor between these points is 0.47 and 1.66.

⁷ <http://en.community.dell.com/techcenter/extras/w/wiki/dvd-store.aspx>

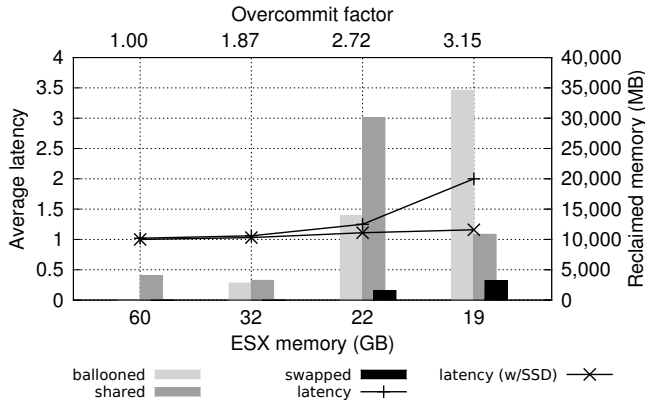


Figure 7. VDI. Average 95th percentile operation latency of fifteen VDI VMs when changing ESX memory from 60GB to 19GB.

However, when ESX memory is reduced to 7GB (2.85 overcommitment), ballooning itself is insufficient to reclaim sufficient memory. Non-trivial amounts of hypervisor-level swapping take place. Owing to the higher wait cost of hypervisor-level page-faults, there is larger performance loss and OPM drops to 8,647.

This experiment was repeated after attaching a 40GB SSD to the ESX Server. The OPM in the presence of the SSD is shown in the same figure. It can be seen that the performance degradation is less in the presence of SSD.

Experiment e: The VDI workload is a set of interactive user-level applications such as *Microsoft Office* suite and *Internet Explorer*. The workload is a custom-made set of scripts which simulate user action on the VDI applications. The scripts trigger mouse clicks and keyboard inputs to the applications. The time to complete each operation is recorded. In one complete run of the workload, 1000 user actions are performed. The measured performance metric is the 95th percentile of latency value of all operations. A smaller latency indicates better performance.

In this experiment, 15 VDI VMs are powered on in the ESX Server. Each VM is configured with 1 vCPU and 4GB memory – the total configured memory size of all VMs being $15 * 4 = 60$ GB. The VMs contain Windows7 OS. Similar to *experiment d*, a memory hog VM, with full memory reservation, is used to simulate overcommitment by effectively reducing the ESX memory on the ESX Server.

Figure 7 shows the 95th percentile of operation latency in the VDI VMs for ESX memory values of {60, 32, 22, 19} GB. At these data points, the memory hog VM has a configured memory size of {12, 40, 50, 53} GB. Owing to the reduced simulated ESX memory, the simulated overcommitment factor (w/ total VM memory size of 60GB) was {1.00, 1.87, 2.72, 3.15}.

The Y-axis (right) shows the total amount of ballooned and swapped memory (MB) from all VMs. The Y-axis (left) shows the average latency of operations in the VDI workload. When ESX memory is 60GB, ballooning and swapping are not observed because 1) page sharing helps reclaim memory and 2) the applications’ total mapped memory and working set fits into the available ESX memory. When ESX memory reduces to 32GB, a total about 3GB memory are ballooned which causes a slight increase in the average latency – 1.06. This indicates that VM’s active working set is being reclaimed at this point. When ESX memory further decreases to 22G and 19GB, the amount of ballooned memory increases dramatically. Also, the swapped memory rises. As a result, the average latency increases to 1.22 and 2 times respectively. This

is owing to the significantly higher page-fault wait cost of ballooning and hypervisor-level swapping.

This experiment was repeated after attaching an SSD, of size 40GB, to the ESX Server. ESX is designed to utilize an SSD as a device for swapping out VM’s memory pages. The average latency in the presence of the SSD is shown in the same figure – 1.00, 1.03, 1.11, 1.16. It can be seen that the performance degradation is significantly lower. This experiment shows that certain workloads maintain performance in the presence of an SSD even when the ESX Server is severely overcommitted.

The experiments in this section show that memory overcommitment does not necessarily indicate performance loss. As long as the applications active working set size is smaller than the available ESX memory, the performance degradation may be tolerable. In many situations where memory is slightly or moderately overcommitted, page sharing and ballooning is able to reclaim memory gracefully without significant performance penalty. However, under high memory overcommitment hypervisor-level swapping may occur, leading to significant performance degradation.

5. Conclusion

Reliable memory overcommitment is a unique capability of ESX, not present in any contemporary hypervisor. Using memory overcommitment, ESX can power on VMs such that the total configured memory of all powered-on VMs exceed ESX memory. ESX distributes memory between all VMs in a fair and efficient manner so as to maximize utilization of the ESX Server. At the same time memory overcommitment is reliable. This means that VMs will not be prematurely terminated or suspended owing to memory overcommitment. Memory reclamation techniques of ESX guarantees safe operation of VMs in a memory-overcommitted environment.

6. Acknowledgments

Memory overcommitment in ESX was designed and implemented by Carl Waldspurger [7].

References

- [1] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using KSM. In *Proceedings of the Linux Symposium*, pages 313–328, 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. Mil6s, D. Murray, S. Hand, and M. Fetterman. Satori: Enlightened page sharing. In *Proceedings of the 2009 conference on 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*, pages 313–328, 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, USA, 2012. ACM.
- [7] C. A. Waldspurger. Memory resource management in VMware ESX server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, Dec. 2002.