ESX Memory Resource Management: Transparent Page Sharing

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

Abstract

Data centers strive to provide maximum compute and storage resources while minimizing cost of operation. To this end, hardware virtualization has become an integral part of data centers. Virtualization provides a tool to effectively and efficiently manage data center resources.

Virtualization software attempts to consolidate multiple virtual machines on to one physical machine. Consolidation ratio is the number of virtual machines placed on a single physical machine. A higher consolidation ratio leads to more savings in cost of operation of the data center.

VMware® ESX® Server is a hypervisor providing efficient consolidation of virtual machines on physical machines. It implements overcommitment techniques to efficiently consolidate virtual machines with more virtual RAM and virtual CPUs than are available on the physical machine.

ESX is said to be memory overcommitted when virtual machines with a combined virtual RAM greater than that available to ESX are powered on. ESX employs different memory reclamation techniques and policies to effectively distribute available physical memory to virtual machines with the goal of maximizing performance.

Transparent page sharing is a method using which ESX collapses multiple memory pages with the same content into a single memory page. The remaining pages become available for use elsewhere. Transparent page sharing works without any modification to virtual machines. It works with all guest operating systems supported by ESX and does not impact performance of guest workloads significantly.

This article describes the transparent page sharing technique. It provides a high-level description of the reclamation technique as well as its low-level configuration.

General Terms memory management, memory reclamation, memory sharing

Keywords ESX Server, memory resource management

1. Introduction

VMware's ESX Server (ESX) is a hypervisor enabling reliable and efficient operation of virtual machines in a data center. Reducing the cost of operating virtual machines in a datacenter motivates placing more virtual machines on a physical machine. To quantify the efficiency of resource utilization, *consolidation ratio* measures the amount of virtual hardware configured on physical hardware. A higher value of consolidation ratio indicates lower cost of operation of virtual machines.

VMware's overcommitment technology is a key enabler for improving efficient utilization of hardware resources. ESX enables overcommitment of physical resources such as CPU and main memory (RAM). Overcommitment enables a user to place virtual machines on a physical machine such that the total configured virtual hardware is more than the available physical hardware, thereby improving the consolidation ratio.

CPU overcommitment enables powering on virtual machines, the total configured virtual CPU (vCPU) cycles of which are greater than the total physical CPU (pCPU) cycles available to ESX. Similarly, memory overcommitment enables powering on virtual machines, the total virtual memory of which is greater than the physical memory available to ESX, called *available ESX memory*.

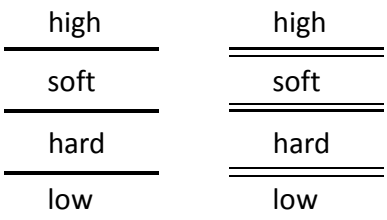
When ESX is memory overcommitted, it must efficiently distribute physical memory pages to different virtual machines in a manner that enables good performance. ESX also attempts to reduce the memory pages consumed by virtual machines when possible. This reduces the actual number of pages consumed by virtual machines, thereby making more memory pages available to other virtual machines. This in turn enables a higher memory overcommitment.

Transparent page sharing (also known as TPS or simply *page sharing*) is a technique using which ESX collapses different memory pages, with the same content, into a single page. When many memory pages are thus collapsed, into a single page, the remaining pages are released. These pages are thereby made available for use elsewhere. They may be allocated to the same or new virtual machine, device drivers or ESX itself.

The page sharing technique is completely transparent to virtual machines. The guest operating system (OS) running inside a virtual machine is not aware that its memory pages are collapsed by ESX. The guest OS, applications executing in the virtual machine, guest device drivers and any other guest component do not require any modification to work with page sharing.

Transparent page sharing benefits all virtual machines running on an ESX Server. Pages can be shared between different virtual machines or even within the same virtual machine. When different virtual machines contain the same or different versions of the same guest OS or applications, there is a high chance of finding common code or data pages between the virtual machines. Page sharing takes advantage of this by finding and sharing the common pages. For example, an ESX Server containing many Windows virtual machines can share common code and data found between the different virtual machines. Similarly, if the different virtual machines contain similar applications – such as the Office suite or web browsers – then code and data pages common to the applications will also be shared. Guest OSes often fill free memory pages with

This article appears as VMware Technical White Paper WP-2013-01E.



(a) (b)

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

the `0x00` pattern. Page sharing helps collapse these *zero*[1] pages by sharing them with a single zero page.

The remainder of this article is organized as follows. Section 2 provides background information about memory reclamation techniques in ESX. Section 3 describes transparent page sharing, Section 4 describes page sharing in the context of *large* pages. Section 5 describes configuration options for controlling page sharing on ESX. Section 6 shows simple experiments to demonstrate page sharing. Section 7 concludes the article.

2. Background and related work

Virtualization is an enabling technology for providing high consolidation ratio in a physical server. Memory overcommitment and *memory reclamation* are integral parts of ESX memory resource management. Memory reclamation reclaims memory from virtual machines and distributes it to those virtual machines that need more memory to perform better.

Related work: The research community has demonstrated page sharing as means of memory reclamation in hypervisors. Singleton [4], uses content based page-sharing on KVM[2], Difference Engine [2], uses sub-page level page sharing with patching in Xen[3] to identify and share pages with similar content. If two memory pages differ only in a small set of byte content, then Difference Engine combines the two pages into a single page and remembers the unique contents where the pages differed. Sartori [3] uses paravirtualized guest and sharing of pages read from storage devices in Xen, KSM [1] in Linux/KVM uses page-sharing to increase memory density of processes and virtual machines.

Background: The memory reclamation techniques of ESX are – page sharing, memory ballooning, memory compression and hypervisor-level memory swapping.

ESX uses different memory reclamation techniques at different times. The memory reclamation method is guided by the available free memory in ESX. ESX defines four different *free memory states* –`high`, `soft`, `hard`, `low` - indicating the amount of free memory available to ESX. Table 1 shows a schematic representation of the memory states of ESX. The actual value of the threshold depends on the available ESX memory. In this table, (a) shows user-visible memory and (b) shows internal threshold to avoid oscillation. Each threshold consists of two sub-thresholds. This prevents oscillation of the ESX free memory state when its free memory lingers near a threshold. The memory reclamation method used by ESX depends on the free memory state of ESX.

[1] A memory page containing only `0x00` content is a *zero* page.

[2] www.linux-kvm.org/

[3] www.xen.org/

Page sharing is an opportunistic memory reclamation method that is active on ESX at all times. ESX continuously searches memory pages for sharing opportunity. Memory pages with identical content across the same or different virtual machines are collapsed into a single new page. The pages that were collapsed are then marked as available for use elsewhere. This method reduces the memory consumption of virtual machines.

Memory ballooning is a memory reclamation technique that entails the service of a guest *balloon driver*. The guest balloon driver is installed in the guest OS. When requested by ESX, the balloon driver allocates guest memory pages from the guest OS. The balloon driver informs ESX about the pages it has been allocated. ESX then considers those pages to be available for allocation elsewhere. These pages remain allocated to the balloon driver which does not perform any read or write operation on them. Memory ballooning is activated when ESX enters the `soft` state and remains active until ESX has moved into the `high` state.

Memory compression and hypervisor-level memory swapping are used by ESX when ESX is in the `hard` or `low` state. Memory compression attempts to compress a memory page owned by the guest OS. If the compression ratio for the given page is good, then ESX keeps the page in a compressed state. If the compression ratio is not good, then the page is swapped out to a suitable swap space. Before attempting memory compression or swapping, on a memory page, ESX opportunistically attempts to reclaim it using transparent page sharing.

The first three memory reclamation techniques – page sharing, memory ballooning, memory compression - are opportunistic in nature. They do not guarantee memory reclamation: a virtual machine may not contain shareable pages, the balloon driver might not be installed or might not be fast enough, the pages of a virtual machine might not yield a good compression ratio. The fourth method, swapping, is a guaranteed method for reclaiming memory. ESX allocates enough swap space to guarantee that, when required, it will always find space to swap out a virtual machine's memory.

The next sections describe transparent page sharing and its configuration controls.

3. Transparent page sharing with small pages

Transparent page sharing works with virtual machines. It also works with *user-worlds*, which are native processes in ESX. This section describes page sharing for virtual machines. For simplicity, it is assumed that virtual machines do not contain any *large* – 2MB sized – pages, that is, all virtual machine pages are *small* – 4KB sized – pages. Page sharing works differently in the presence of large pages, described later.

The goal of transparent page sharing is to find all virtual machine pages with an identical content and replace all of them with a single page with the same content. All entities that were using those pages now point to the new page. The new page is read-only and is marked as *copy-on-write*. Whenever any entity attempts to write to the page, a private copy of that page is made for that entity before allowing the write operation to proceed.

Table 2 shows two virtual machines that contain pages with identical patterns between them. There are two sets of pages with identical patterns. Identical content is shown with the same shade of gray. All other pages contain unique content. After sharing has taken place, one page of each content type is created. The virtual machines now point to these shared pages from those locations where they were earlier located in the virtual machines.

Figure 1 describes the steps performed for comparing and sharing two pages with identical content.

In Figure 1 (a) and (b), ESX continuously scans memory pages belonging to virtual machines. Every second, ESX selects a set of n small pages from a virtual machine. This set of pages, P, are pages

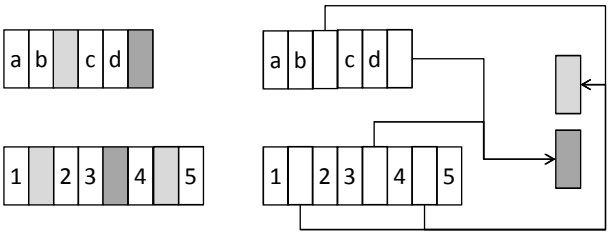


Table 2. Left: Two virtual machines with two sets of pages with identical content. **Right**: Pages with identical content are replaced with a common read-only page with the same content.

that are not already shared, compressed, ballooned or swapped. These pages are candidates for sharing.

In Figure 1 (c), ESX computes a hash, `H(p)`, from the content of each page, p. Creating a hash provides ESX with an easy method for comparing the content of two pages. These hashes are stored in a hash table, `T`, shown in Figure 1 (d). For each computed hash, `H(p)`, ESX searches the hash table to see if another entry was already present in it.

If a matching hash `H(p')` is found in the table, shown in Figure 1 (e), for another page p', then the pages p and p' have a high probability of having the same content. ESX now compares the full content of p and p' to determine if their contents are the same. If they have the same content, then the content of these pages are replicated in a third new page and these two pages are released. The virtual machines to which p and p' belonged now point to the new page with read-only access. If their contents do not match, shown in Figure 1 (f), then p and p' are not shared. However, the computed hash for p is stored in the hash table as a *pshare hint*. This is done so that it become possible to compare p with another page in a future iteration.

When two pages are collapsed into one page, one page is effectively released. It is possible for many pages to be shared with a single page, if their contents match. Hence, when virtual machines contain memory pages with repeated patterns, the total amount of saved pages can be quite high. For example, if $1,000$ guest memory pages, among all powered-on virtual machines, contain the *zero*-pattern, then as many as 999 pages can be saved.

When a virtual machine contains a page that has been shared with another page, other page with the same content might belong to another virtual machine, V. This does not permit one virtual machine to determine or infer the contents of other pages in V. Complete data and execution isolation is always ensured between all virtual machines which are sharing pages with the same content. A virtual machine is also never aware that any of its memory pages have undergone page sharing. Page sharing is a completely transparent process.

A memory page that has undergone sharing is marked as *read-only*. A virtual machine may read from this page at any time. However, if a virtual machine writes to that page, then the content of that page will change. This change will become visible to other virtual machine. To avoid this unwanted event, ESX makes a private copy of the shared page when a virtual machine attempts to write into that page. The private copy is given to the virtual machine attempting the write operation. Other virtual machines referring to the shared page will continue to see the unchanged content.

Figure 2 shows the steps performed by ESX when a virtual machine writes to a shared memory page. In Figure 2 (a) three virtual machines contain a page each, with a common pattern. A shared read-only page backs the shared content. All virtual machines have

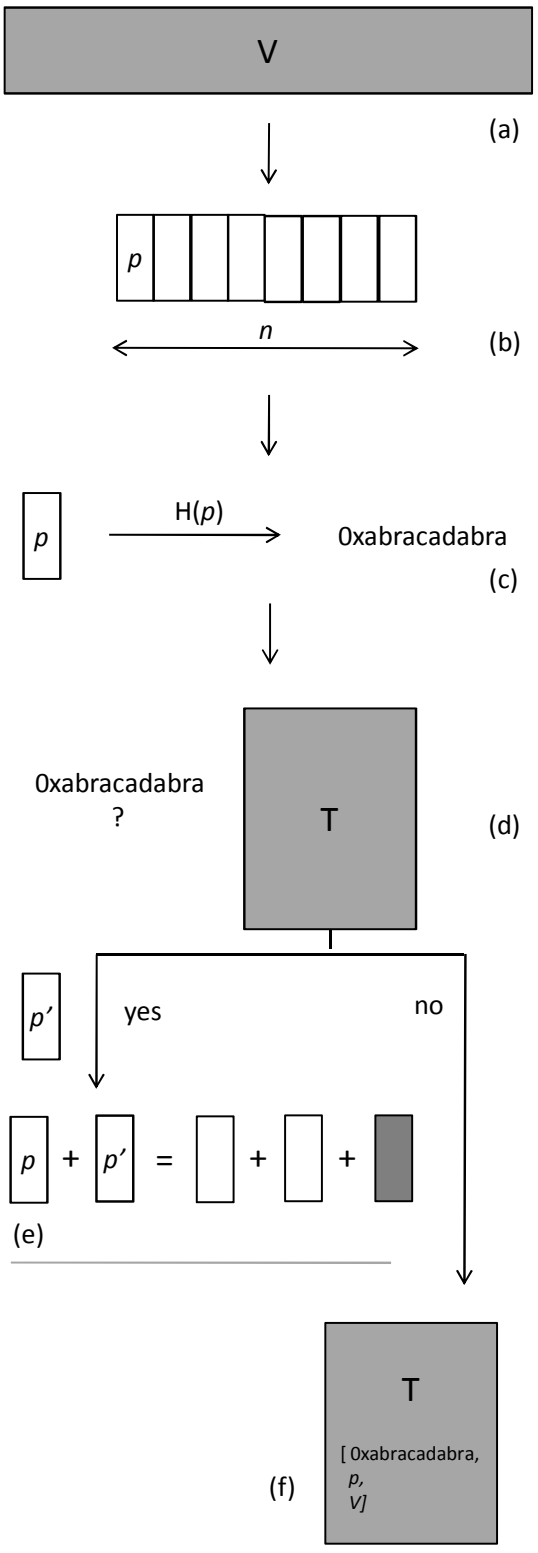


Figure 1. Steps for finding and sharing pages with identical content. (a) A virtual machine, V, is being scanned (b) n pages from V are selected (c) For each selected page, its hash H is computed (d) The hash is searched in a table (e) If matching hash is found, for another page p' that is not shared yet, then p and p' are replaced with a common page (f) If not found, H is stored in the table.

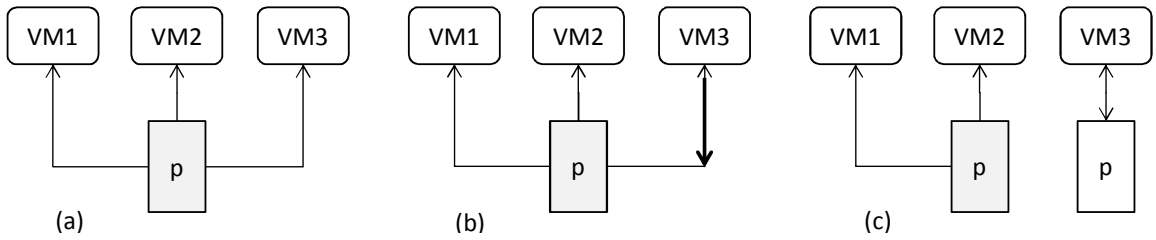


Figure 2. Steps performed when a virtual machine attempts to write into one of its pages that is being shared. (a) Initial state where 3 virtual machines are sharing a page with a specific pattern. All virtual machines have shared read-only access. (b) VM3 attempts to write to the shared page. (c) The write operation is trapped, a private copy of the page is made for VM3 and VM3 given read-write access to the private page.

shared read-only access. In Figure 2 (b), VM3 attempts to write into the shared read-only page. In Figure 2 (c), ESX makes a private copy from the shared page for VM3 and allows VM3 read-write access to the private page. VM1 and VM2 continue to have read-only access to the shared page.

Hashing memory page content and comparing with other pages is a computationally expensive operation. ESX performs this continuously for each powered-on virtual machine, as long as that virtual machine is powered on. In order to limit excessive CPU consumption, ESX limits the value of n – the number of pages scanned every second from each virtual machine. The value of n, varies from one virtual machine to another. It depends on the configured memory size of the virtual machine. Virtual machines with more configured memory size have a higher value of n.

Sharing pages is a continuous process. Not all possible shareable pages in a virtual machine are shared together or shared when that virtual machine first contains shareable content. A shareable page might get selected by ESX at some time during the virtual machines lifetime. It will get shared at that time. Hence, if a guest OS or application generates shareable memory content, it might take some time until ESX comes around to share that page and save memory. In addition to continuous scanning, ESX opportunistically attempts to share a page before attempting to compress or swap it out during memory reclamation in `hard` or `low` memory states.

This section describes sharing memory pages of size 4KB. The next section describes the steps performed by ESX in the presence of large – 2MB sized – pages.

4. Transparent page sharing with large pages

The previous section describes how ESX finds and shares guest memory pages, in an attempt to reduce the total memory consumption of virtual machines. The guest memory pages were assumed to be small pages (4KB sized). This section describes page sharing for virtual machines in the presence of large pages.

Modern processors and guest OSes provide support for large memory pages, for example, 2MB or 1GB. Empirical studies have shown that the likelihood of finding shareable content decreases with the increase in the size of the page being considered for sharing. For example, the chance of finding two pages with identical content is higher when considering two 4KB pages than when considering two 2MB pages. Therefore, ESX restricts transparent page sharing to work with small guest memory pages.

A difficulty presents itself when virtual machines are configured to use large pages in ESX. Large pages in ESX are 2MB sized pages. Using large pages in ESX has certain advantages – 1) reduced state maintained by ESX in machines using EPT[4]/NPT[5] based hardware virtualization 2) improved memory access latency for virtual memory 3) fewer TLB[6] misses in the CPU because of reduced page table levels. Because of these advantages, guest OSes and workloads requiring high performance work well with large pages. In fact most guest workloads perform better when ESX is configured to use large pages.

However, as a result of using large pages, virtual machines might potentially be deprived from transparent page sharing and its benefits of saving memory. ESX does provide the facility where virtual machines can take advantage of large pages as well as benefit from transparent page sharing of small pages.

As mentioned in Section 3, ESX continuously scans the memory pages of a virtual machine for sharing opportunity. During this scan, ESX also scans 4KB sized pages which together constitute a 2MB sized large page. However, instead of attempting to share the 4KB sized constituent page, ESX simply computes its hash and stores it in the hash table as a *pshare hint*. Essentially, the steps from Figure 1 (a), (b), (c) and (f) are performed.

When the memory state of ESX falls below `high`, then ESX selects a subset of large pages and converts them into small pages. These small pages can then be scanned for sharing, compressed or swapped out to the virtual machine's swap space. The *pshare hints* stored in the hash table, by scanning 4KB pages within a 2MB page, now help identify shareable 4KB pages which earlier constituted a 2MB large page.

Therefore, when a virtual machine is using large pages, transparent page sharing will become effective only when the memory state of ESX falls below `high`. Even though the virtual machine may contain shareable content in the form of small pages, these will be considered for sharing only at this time.

5. Page sharing configuration

Transparent page sharing is always enabled on ESX. ESX allows the user to control certain aspects of page sharing. These controls are provided from the *Advanced configuration options* under the section `Mem`.

The configuration options are described here.

ShareScanTime Estimated time in minutes within which to scan all memory pages of a virtual machine for page sharing oppor-

[4] Extended Page Tables technology of Intel Corporation

[5] Nested Page Tables technology of Advanced Micro Devices

[6] Translation Lookaside Buffer

tunity.

ESX computes a scan *rate* for each virtual machine. This is the number of pages to scan every second for page sharing. The scan *rate* is derived by dividing the configured memory size of the virtual machine by `ShareScanTime`. Hence, a lower value of this configuration options yields a higher rate. The default value is 60 minutes.

$$rate = memsize/(ShareScanTime * 60) \quad (1)$$

Where:

rate	- pages per second
memsize	- pages
`ShareScanTime`	- minutes

The scan *rate* is calculated for every powered-on virtual mchine. Its value depends on the configured memory size of the individual virtual machine and the global configuration option `ShareScanTime`.

ShareRateMax Upper bound on the value of *rate* computed in Equation 1.

Virtual machines with a large configured memory size will result in a high value for *rate*. A high scanning rate might in turn result in high CPU consumption for page sharing. The CPU resources spent in sharing pages are accounted toward the CPU resources allocated to a given virtual machine. If a significant amount of CPU resource is spent on page sharing, then this will reduce the CPU resources allocated to the guest OS.

In order to limit CPU consumption on a per-virtual machine basis, an upper bound on the value of *rate* is provided. The default value is 1024.

ShareScanGHz Mega-bytes of memory pages to scan per second for each GHz worth of CPU resource available to ESX.

This limits the total CPU resources being consumed on an ESX Server for page sharing. Since page sharing is a continuous activity, it consumes system bus resources. If an ESX Server has many virtual machines, then the total amount of CPU and memory operations for page sharing might be quite large.

In order to limit global CPU and memory operations for page sharing, this configuration option places an upper bound on global page sharing activity.

The default value of this option is 4. Hence, on a 8-core system with 3GHz per core, the total page scanning rate will be limited to $8 * 3 * 4 = 96$ MB/sec ($24,576$ pages/sec). If the total scanning rate of all powered-on virtual machines exceed this value, then all scanning rates are scaled by an appropriate multiplier.

ShareIncPct Percentage auto-increase of per virtual machine page scan rate when many sharable pages are found.

ESX automatically raises the page scanning rate, if it finds that page sharing is yielding good memory reclamation. This configuration option is the percentage by which ESX increases the per virtual machine value computed in Equation 1. The increases value is subject to the bounds described earlier. The default value is 100.

ShareDecPct Percentage auto-decrease of per virtual machine page scan rate when many sharable pages are not found.

ESX automatically lowers the page scanning rate, if it does not find enough pages for sharing. The default value is 50.

This section shows parameters for controlling the behavior of transparent page sharing.

6. Experiments

The previous sections have described how page sharing looks for an opportunity to reclaim guest memory at all times. This section demonstrates the behavior of page sharing with simple experiments.

All experiments are conducted on a development build of ESX 6.0. The ESX Server has 32GB RAM, 8-core AMD configured in 4-NUMA node configuration. The virtual machines used in the experiments contain RHEL6 OS. Ballooning has been disabled in these experiments, in order to simplify the explanation. Page sharing has been configured with a `ShareScanTime` of 10 and `ShareRateMax` of 7168, while all other parameters are set to the default value.

The ESX Server has 32GB memory and consumes about 2GB overhead memory after the virtual machines are powered on. Thus the total free memory available to ESX is about 30GB. Therefore, ESX will be memory-overcommitted when the total configured memory of powered-on virtual machines exceed 30GB. Detailed calculation of memory overhead is omitted for simplicity.

6.1 Small pages

The effect of page sharing on shareable pages of a virtual machine is shown here. In this experiment, ESX is configured to allocate only small pages to the virtual machine. Note, that the default behavior of ESX is to allocate large pages to a virtual machine when possible.

Under-committed Figure 3 (a) shows one RHEL6 virtual machine – configured memory size = 16GB, no memory limit, no memory reservation – being powered on in the ESX Server. Since the ESX Server has 32GB RAM, it remains under-committed at all times in this experiment. ESX uses small pages to back the virtual machine's memory.

At the time marked by **1** the guest OS has completed booting. At this point, the virtual machine consumes about 980MB of memory. At **2**, a workload is started in the virtual machine. The workload allocates 15GB of guest memory and writes the pattern `0xff` to all the allocated memory pages. At **3**, the workload has complete allocating memory. At this point, the virtual machine consumes about 16GB of memory. After writing the pattern is complete, the workload is terminated.

Since the page content of the virtual machine is predominantly the pattern `0xff`, a large number of the virtual machine's pages are shareable with each other. Page sharing is active at all times. From point **3** onward, the total memory consumption of the virtual machine starts decreasing continuously. At the same time, the total guest pages being shared rises at the same rate. At **4**, the experiment is terminated when the total shared pages and total consumed pages become anecdotally equal. If the experiment were allowed to continue, then all shareable pages in the virtual machine would have been eventually shared.

This experiment shows that ESX continuously searches for shareable small pages in powered-on virtual machines and shares them.

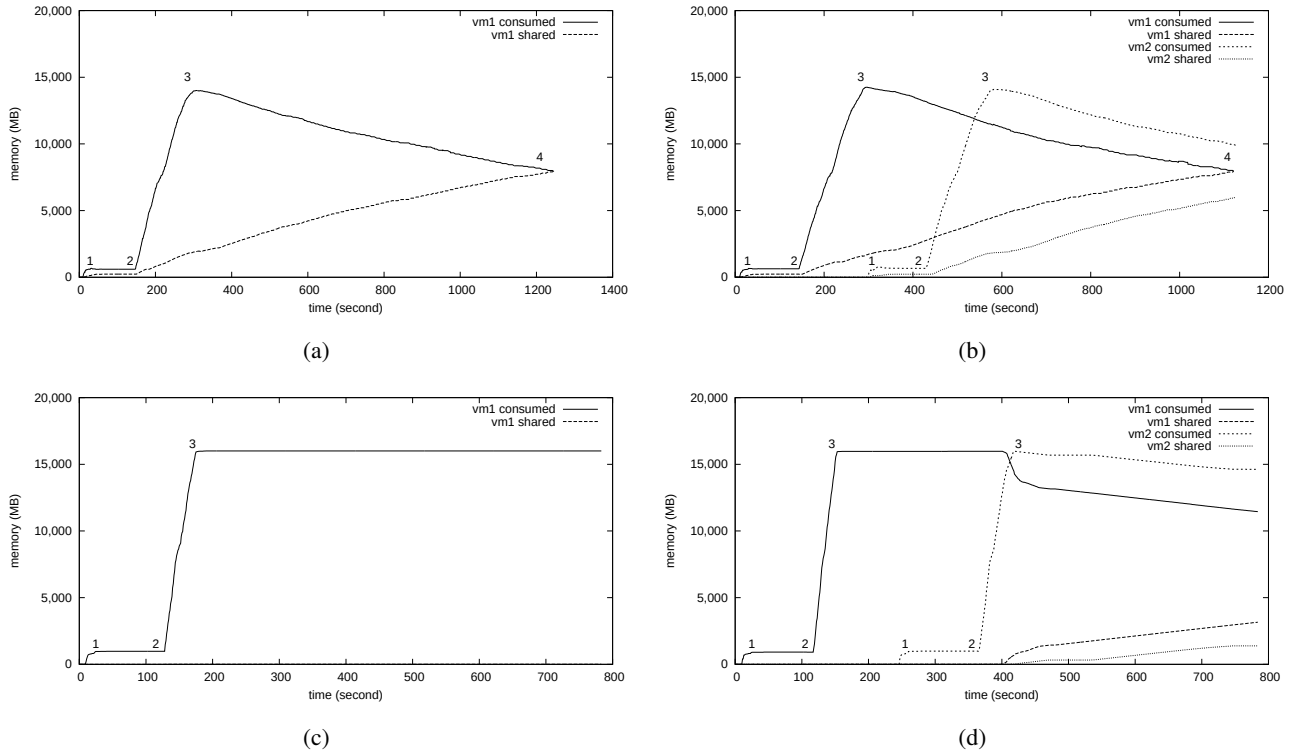


Figure 3. Experiment showing page sharing with 1 and 2 RHEL6 virtual machines executing a memory bound workload. (a) 1 virtual machine where ESX uses small pages to back virtual machine's memory. (b) 2 virtual machines where ESX uses small pages to back virtual machine's memory. (c) 1 virtual machine where ESX uses large pages to back virtual machine's memory. (d) 2 virtual machines where ESX uses large pages to back virtual machine's memory.

Over-committed Figure 3 (b) shows two RHEL6 virtual machines – configured memory size = 16GB each, no memory limit, no memory reservation – being powered on in the ESX Server with an interval of 400 seconds between them. ESX uses small pages to back virtual machine's memory. ESX is overcommitted in this experiment since the total configured memory size of powered-on virtual machines is 32GB (exceeds 30GB).

Points **1** through **4** in this experiment are similar for both the virtual machines and bear the same interpretation as in the previous experiment.

In this experiment, when each virtual machine powers on, it boots RHEL6. A workload inside the virtual machine consumes 15GB of memory and writes the pattern `0xff` to it. ESX continuously searches for shareable pages and shares them. It can be seen that the consumed memory for each virtual machine decreases continuously, while the number of pages shared in each virtual machine rises at a corresponding rate. Although not shown in the figure, memory reclamation using hypervisor-level swapping is avoided, since page sharing reclaims sufficient memory from the virtual machines.

This experiment demonstrates that page sharing continues as long as a virtual machine remains in the powered-on state, regardless of whether ESX is memory overcommitted or not.

6.2 Large pages

The interaction of page sharing with large pages is shown in this section. In this experiment, ESX is configured to allocate large pages to virtual machines whenever possible. This is the default behavior of ESX.

Under-committed A large page in ESX is a memory page of size 2MB. ESX allocates large pages to virtual machines by default. Only if there are no free large pages with ESX, will it allocate a small page to the virtual machine.

Since page sharing can share pages of size 4KB, it will not share large pages in a virtual machine. However, it will continue to scan 4KB sized constituent small pages in 2MB sized large pages and store pshare hints in the hash table (see Section 4).

Figure 3 (c) shows one RHEL6 virtual machine – configured memory size = 16GB, no memory limit, no memory reservation – being powered on in the ESX Server. ESX uses large pages to back the virtual machine's memory. The ESX Server is always in an undercommitted state since its total memory size is 32GB.

This is similar to the single virtual machine experiment in Section 6.1. In this case, the events marked by **1**, **2** and **3** are similar. However, since the virtual machine has been allocated only large pages and ballooning or swapping is not required, the large pages do not get converted into small pages. Hence, page sharing does not reclaim any memory pages from the virtual machine.

The peak memory consumption in Figure 3 (a) is lower than in (c). This is because in the former case, ESX was continuously sharing pages while the workload was allocating and writing into memory. In addition, creation of large pages by ESX for a virtual machine might result in slightly higher memory consumption.

This experiment shows that ESX does not share memory, that is backed by large pages, in a memory-undercommitted state.

Over-committed Figure 3 (d) shows two RHEL6 virtual machines – configured memory size = 16GB each, no memory limit, no memory reservation – being powered on in the ESX Server with an interval of 400 seconds between the two power-on operations. ESX is overcommitted when the second virtual machine is powered on. ESX uses large pages to back the virtual machine's memory.

When the first virtual machine is powered on and allocates and writes to its memory, the behavior is similar to that in the previous experiment. Points **1**, **2** are similar to the previous experiment. No page sharing takes place in this virtual machine until the second virtual machine has powered on and its workload has allocated a large portion of its memory. At point **3** of `vm2`, ESX dips below the `high` state, since both the virtual machines have consumed about 30GB memory. At this point, ESX starts reclaiming memory from both virtual machines. It selects a subset of allocated large pages from both virtual machines and converts them into small pages. Some of the small pages are swapped out. The remaining small pages are progressively shared by ESX.

This experiment demonstrates that ESX does not share content from large pages when a virtual machine's pages are backed by large pages. However, when overcommitted and under memory pressure, ESX will proceed to reclaim memory from virtual machines by means of ballooning and swapping. At this time, ESX will convert large pages into small pages and will also take advantage of page sharing to reclaim memory.

The experiments in this section show that ESX reclaims memory by means of page sharing when virtual machine pages are backed by small or large pages. Reclamation is continuous in the absence of large pages and reclamation in the presence of large pages is delayed until ESX is under memory pressure.

7. Conclusion

This article provides a description of transparent page sharing in ESX. Transparent page sharing is an opportunistic technique, employed by ESX, for reclaiming memory from virtual machines. Using this technique, ESX is able to collapse memory pages, with identical content, between one or multiple virtual machines. It works transparently, that is, guest OSes and workloads do not need any modification to work with this technology. Page sharing benefits all guest OSes and all workloads and is an effective method for increasing the consolidation ratio in an ESX Server.

8. Acknowledgments

Memory overcommitment in ESX was designed and implemented by Carl Waldspurger [5].

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] G. Miłós, D. Murray, S. Hand, and M. Fetterman. Satori: Enlightened page sharing. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 1–1. USENIX Association, 2009.

[4] 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.

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