The Yin and Yang of Memory Overcommitment in Virtualization

The VMware vSphere 4.0 Edition

YP Chien, PhD, CISSP

Introduction

Yin & Yang is an ancient Chinese belief about the existence of two complementary forces in the universe — Yang represents everything positive or "bright" and Yin stands for the opposite or everything negative or "dark." Everything has both Yin and Yang aspects and we can't ignore one or the other.

We see the same Yin & Yang effects in virtualization. For example, we can look at consolidation ratios; the higher the consolidation ratios, the better the server utilization, reduced power consumption and hence fewer physical servers to manage. On the other hand, the indiscriminate quest for higher consolidation ratios can cause performance problems and worse, such as increasing the business risks due to hardware failure.

One of the key tools to drive higher consolidation ratios is to "oversubscribe" or overallocate computing resources such as system memory. With a new generation of processors that sport 6, 8 or more cores and the capability of supporting 20 or more virtual machines in a VMware vSphere ESX host, the limiting factor for scaling up further is now physical memory. Overallocating memory resources could eventually cause *memory overcommitment,* which now plays a greater role than ever in driving up consolidation ratios.

Memory overcommitment in this guide means that the total memory size consumed by all running virtual machines exceeds the total size of the physical server memory. VMware ESX provides memory allocation and management capabilities to achieve efficient memory utilization while maintaining memory performance. Still, there will be instances where the total VM memory requirements may sufficiently exceed available physical memory to negatively impact workload performance.

In this whitepaper, we will use controlled tests to demonstrate how VMware ESX manages memory while under memory pressure, and see how ESX strikes a balance between VM performance and memory reclamation. We will also show potential problems that could impede performance when memory requirements greatly exceed available physical memory.

Memory Management in VMware ESX server

Dynamic Memory Allocation

ESX dynamically manages the amount of physical memory allocated to each VM based on:

- VM usage or need
- System load
- Priority

While a VM assumes it can access all of the memory that is configured, ESX allocates memory to the VM only when it is actually needed.

ESX continues to allocate additional memory to a VM as long as there is sufficient physical memory available to satisfy the VM workload requirements. But what happens when there is a memory shortage and not enough free memory to satisfy workload demands from the VM? When memory contention arises, ESX will reclaim unused memory (***Idle Memory***) from VMs that have less need and redistribute it to VMs that require more memory. With scarce memory resources, memory pages will be reclaimed, preferentially from VMs that are not actively using their memory allocation or from VMs that have low memory utilization. If we factor out utilization from this scenario, ESX will allocate memory based on priority as defined by Shares.

Dynamic Memory Reclamation

ESX utilizes the following methods in sequence to reclaim idle memory from VMs when it is under memory pressure:

- Memory Ballooning
- Memory Compression, new in vSphere 4.1
- VMkernel Swapping

Under memory pressure, ESX prefers memory ballooning over VMkernel swapping to force VM to give up memory pages and let them use their own memory management processes to rebalance memory requirements. When memory ballooning is not possible or insufficient to meet the memory needs, in vSphere 4.0, ESX utilizes VMkernel swapping as the last resort. ESX reclaims VM memory by paging out memory to a swap file on disk storage, without any guest VM involvement. This swapping will certainly impact VM performance, especially in cases where this swapping causes a VM to further increase Windows page swapping.

For details on how memory ballooning and VMkernel swapping work, please refer to:

http://www.kingston.com/branded/pdf_files/Final_esx3_memory.pdf.

In the following sections, we will show Memory Ballooning and VMkernel Swapping in action and how they interact with each other to ease ESX memory pressure. We will also examine their associated performance penalties and show why Swapping must be avoided if possible. In vSphere 4.1, instead of swapping, ESX will store the swapped out pages in a compression cache if these pages can be compressed, which in effect avoids the long latencies caused by swapping to disk. Since our test environment is based on VMware vSphere 4.0, the new Memory Compression feature will not be covered here but will be covered in a future white paper.

ESX Memory Reclamation States

ESX dynamically adjusts memory allocations when changes in the amount of free memory cross predefined thresholds. ESX uses four thresholds to reflect different reclamation states: ***high***, ***soft***, ***hard*** and ***low***, in which the thresholds default to 6%, 4%, 2%, and 1% of server memory, respectively. The current ESX memory state can be viewed by running the esxtop utility.

ESX should generally be in the ***high*** memory state, indicating free memory is sufficient and no reclamation is needed. With sufficient memory resources, VMs can continue to request memory and keep their allocated memory.

When ESX is in the ***soft*** state, ESX starts to reclaim memory by first using memory ballooning, and in the case of vSphere 4.0 and earlier, resorting to VMkernel swapping only in cases where ballooning is not possible or insufficient to reclaim enough memory. In both the ***hard*** state and the ***low*** state, ESX must resort to VMkernel swapping to forcibly reclaim memory.

Since the memory ballooning reclaiming processes take time to execute and might not respond fast enough to address transient memory pressures, VMkernel swapping can still occur as a result. We will later show in our experiments that ESX started memory reclamation before it reached the ***soft*** state, and initiated memory ballooning and even VMkernel swapping while in the ***high*** memory state. This scenario may be due to the fact that the ESX memory state reporting lags the actual real time memory condition that ESX sees and reacts to address.

Where is all the VM memory?

The efficiency of VM memory utilization will be used by ESX to find preferential candidates for memory reclamation. We will show using the following example to illustrate how we can determine the *active,* or *working set,* memory, *idle* memory, *allocated/consumed* memory and finally *configured* memory for a VM. This will help us better appreciate ESX's memory reclamation strategies and see how idle memory is involved to determine VMs' memory utilization efficiency.

The VM memory usage can be best illustrated by the following screenshot of the vSphere client in the "Hosts and Clusters" view with the *usyp-win3k003* VM selected, and by selecting the *Resource Allocation* tab :

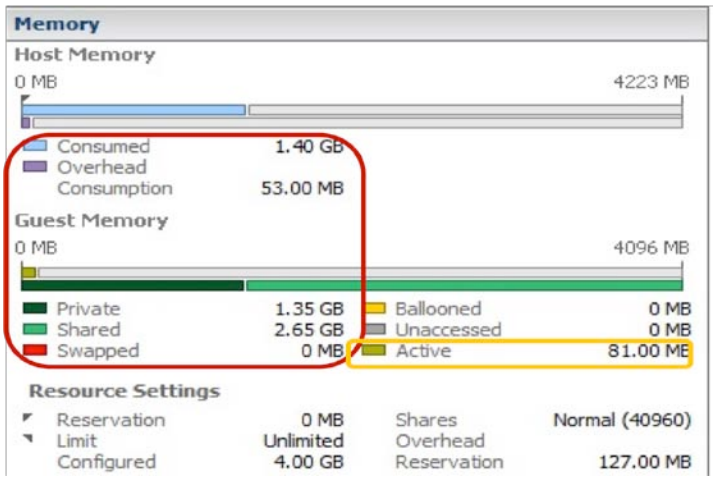


Figure 1. Memory Resource display in vSphere client

Figure 1 shows that the *usyp-win3k001* VM has 4096MB of *Configured* memory, which was set when the VM was created. The *Host Memory* of 4223MB is actually the amount of memory that will be allocated to this VM, which includes the 4096MB of configured memory plus 53MB of overhead memory.

The total *Consumed* memory of 1.4GB as shown in Figure 1 is the sum of 1.35 GB of *private* memory plus 53 MB of *Overhead Consumption*. All of the 1.4GB *Consumed* memory in this case is backed by ESX with physical memory.

The *Shared* memory of 2.65 GB in Figure 1 results from boot time optimizations in vSphere. Windows OS zeroes the memory assigned to it during boot time as part of the initialization process. vSphere shares most of such zeroed pages resulting in significant memory savings. Later, when windows OS or the applications in the VM writes to the memory pages that were previously zeroed, the page sharing is broken. Furthermore, the *Active* memory of 81MB is the amount of memory activity sampled and detected by the ESX memory management system. The amount of memory that is *actively* utilized vs. the amount that is *consumed* by the VM will determine what ESX terms as *idle* memory.

The esxtop utility can also be used to obtain the same memory usage statistics shown in Figure 1 for *usyp-win3k001*. Figure 2 shows the memory usage, memory ballooning and VMkernel swapping statistics for the running VMs:

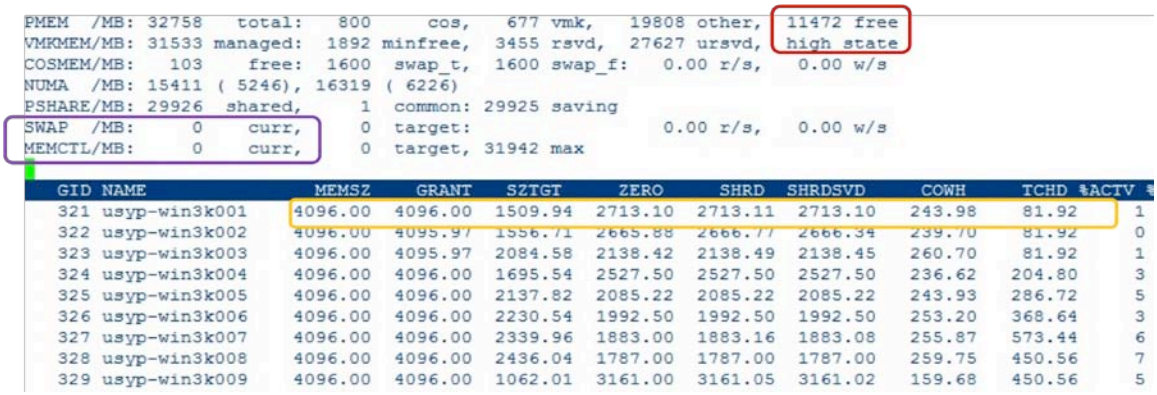


Figure 2. Memory usage & ballooning statistics from esxtop

In the top right corner of Figure 2, we can see that the ESX is in ***high*** memory state with over 11 GB of free physical memory. Furthermore, there were no memory ballooning and/or VMkernel swapping activities as indicated by the MEMCTL and SWAP statistics, which show zero values.

To help interpret the esxtop memory statistics as related to VM memory usage, we note that all Windows Operating Systems zero all their memory pages during the boot process. ESX recognizes this zeroing mechanism and doesn't allocate physical memory to these pages. Since these pages were zeroed out by the guest OS anyway, ESX pools all of them under *shared* pages (**SHRD)**, and the *shared saved* (**SHDSVD**) statistics. These statistics only indicate that the memory pages should have been allocated to specific VM but were not as the VM did not need them yet. So, the actual amount of physical memory ESX allocated to the *usyp-win3k001* VM is the difference between the *granted* memory (4096 MB - the **GRANT** statistics) and the *zeroed* memory (2713 MB- the **ZERO** statistics), which results in *allocated* memory of 1383MB or 1.35GB, the same number shown in Figure 1 as *private* memory.

ESX Memory Ballooning and Swapping under the hood

With memory overcommitment, we can aggressively drive up consolidation ratios and hope that TPS, memory ballooning and, if necessary, VMkernel swapping bail us out during a peak utilization period. However, with excessive memory overcommitment, ESX will be forced into aggressively reclaiming memory, possibly starving VM for memory and forcing the VM Operating Systems to resort to paging. When this happens, VM's performance will likely be negatively impacted.

To illustrate the potential impacts of memory ballooning and VMkernel swapping on performance, we performed several memory load experiments on a HP ProLiant® DL 360 G6 server with 32GB of Kingston® server memory and two Intel Xeon E5540 quad-core processors. All tests were conducted using 12 Windows 2003 VMs provisioned through the same VM template configured with 4GB of memory.

The testing objectives were to:

- Examine ESX dynamic memory reallocation in action and understand how memory ballooning, VMkernel swapping and TPS impact the performance of each individual VM and the overall system.
- Show why memory configuration using reservations or limits could help improve performance on an individual VM but may have adverse effects on other VMs and the overall system.

The Memory Load Generator Utilities

In order to simulate high memory demands that could force memory ballooning and VMkernel swapping, we developed a set of utilities and scripts to generate workloads with varying memory sizes and load duration requirements. To help track and analyze the impact on VM performance, we also developed scripts to extract memory counter logs from the esxtop and Windows Perfmon utilities.

The Memory Load Generator utilities put together for this article consist of the following components:

- Memory Load Generator utilities – a set of Windows executables that generate workloads with configurable memory sizes and load duration
- PowerShell scripts with VMware PowerCLI snapins that invoke and remotely run the memory load utilities in the Windows 2003 VM
- PowerShell scripts with VMware PowerCLI snapins that power-up, power-down and vMotion any of the 12 Windows 2003 VMs remotely (without using the interactive vSphere client program).

We designed the test components to make them:

- **Configurable** – The Memory Load Generator program is fully configurable. This allows us to exercise ESX memory management functions with high transient memory demands of our choosing.
- **Manageable** – We can execute programs remotely and without having to interactively log into the VM under test. Furthermore, the PowerShell scripts can communicate to the vSphere vCenter server (through the PowerCLI snapins) to remotely execute any program in a VM.
- **Measureable** – With vSphere 4.0 and later, VMware performance counters are included in the Windows Perfmon tool (when vmware-tools is installed). This allows for a performance view both internally and externally to the Windows VM. Furthermore, the PowerShell scripts can start/stop both the esxtop and the Windows Performance monitor to collect memory performance counters for offline analysis. The details on using esxtop and Windows Performance monitor in batch mode, and all their associated counters are beyond the scope of this white paper.
- **Repeatable** – All memory workload generation and log collecting steps are scripted and executed as PowerShell scripts to reduce manual data collection errors.

To show the flexibility of this Memory Load Generator System, here is an example of how we generate an arbitrary workload with memory size of 500 MB with the following timings:

- Ramp up the memory load of 500MB request in 60 seconds
- Generate a non-zero CPU load
- Maintain the workload with 2 minutes of high memory consumption activity to simulate the memory usage or low idle memory
- Exit the memory load generator program, which causes the memory to be released back to the OS

The command syntax is as follows:

*LoadStart.bat <memory load size in MB> <%CPU load> < Memory Activity - **N**, **L**, **M** or **H**> <Ramp up speed in 3-sec increments> <total elapse time in seconds>*

To simulate the above workload, we will run the following command, on the *usyp-win3k09* VM:

LoadStart.bat 500 20 H 20 180

Figure 3 shows the Windows *Memory Available Mbytes* counter (using Perfmon), which is being collected as the memory load program is being executed, showing the 500MB reduction in the available memory during the duration of the test:

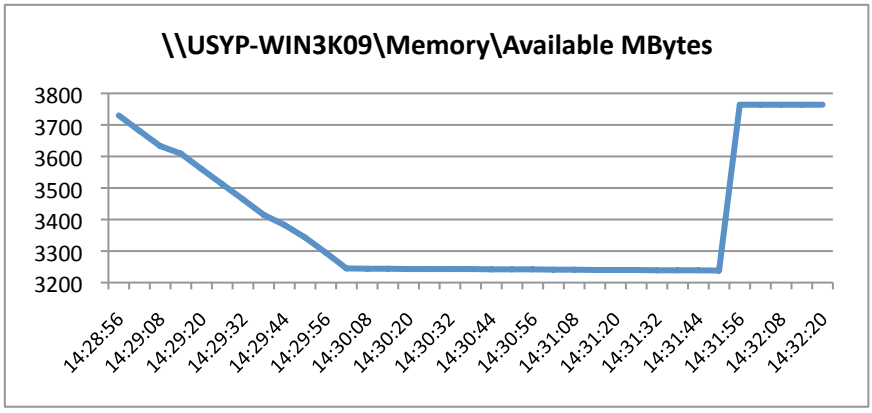


Figure 3. Example of running the Memory Load Generator

ESX Test System Initial Load and VM Memory Activities

We ran the tests with 12 identically-configured Windows 2003 VMs, each with 4GByte of configured memory on the ESX server, which results in 60% of memory overcommit as shown in Figure 4b. We employed the Memory Workload Generator utility to produce a total of 21GB of memory usage on the ESX server, and with about 11 GB of free memory as shown in Figure 4a.

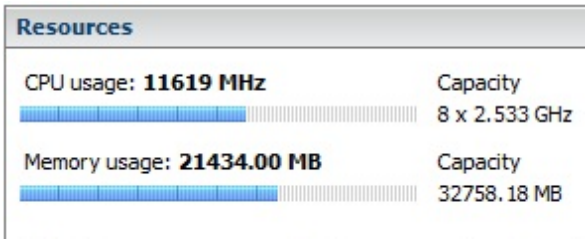


Figure 4a. Memory Usage before starting load test

Using the load generator, we provisioned the first three VMs, *usyp-win3k001* to *usyp-win3k003*, with no memory activity (-N option) to simulate high idle memory. We shall see in our test results in the later sections that these three VMs will be the first targets for memory reclamation by the ESX during a memory shortage. The next 5 VMs, *usyp-win3k004* to *usypwin3k008*, were configured with medium memory workload activities so that we could examine the potential impact of idle memory under memory pressure. We configured the last 4 VMs, *usyp-win3k009* to *usyp-win3k0012* to simulate high transient memory demands that would force ESX into memory pressure.

The resulting memory load and each VM's memory activities are shown in Figure 4b, which utilizes esxtop utility statistics. The esxtop MEM overcommit average statistics shown on the first line of Figure 4b indicates that there are 0.59 or 59% of memory overcommit level in the last 1 minute, 5 minutes and 15 minutes, respectively.

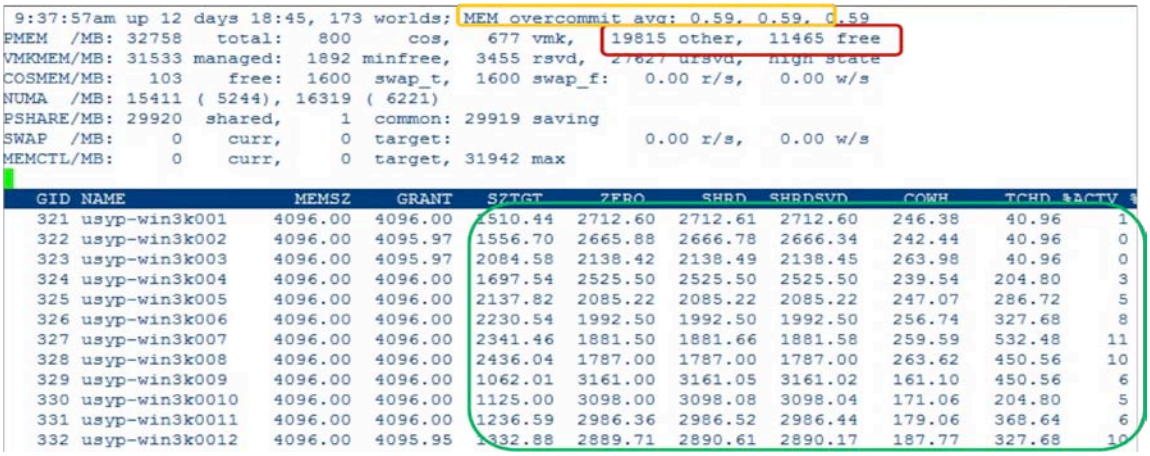


Figure 4b. Memory Activities before starting test

To simulate the effects of memory pressure, we ran 4 additional memory workloads totaling 9.5GB for a period of 12 minutes on top of the four VMs as follows:

VM	Memory Load Size - MBytes	Load Pattern
usyp-win3k009	2,500	Starts immediately and lasts 11 min.
usyp-win3k0010	2,500	Starts immediately and lasts 12 min.
usyp-win3k0011	2,000	Starts one min. later and lasts 8 min.
usyp-win3k0012	2,500	Starts 4 min. later and lasts 5 min.

The deliberate delay to initiate the last two workloads and the slow ramp up of the last workload on *usyp-win3k0012* allows us to capture enough log data to study how ESX manages its dynamic memory reallocation during a memory shortage. The aggregate workload of 9.5GB should starve ESX memory and drive ESX free memory close to the 6% ***high*** memory threshold. At such time, we would expect ESX to start memory ballooning and even VMkernel swapping to reallocate memory. The result of the simulated memory load "storm" patterns for each of the 4 VMs (which is obtained by calculating the memory reductions caused by the memory load using the VM's Windows "Available Memory" Perfmon counter) is shown in Figure 5. The "Total Memory Loads" shown is the sum of the memory workloads for the 4 VMs.

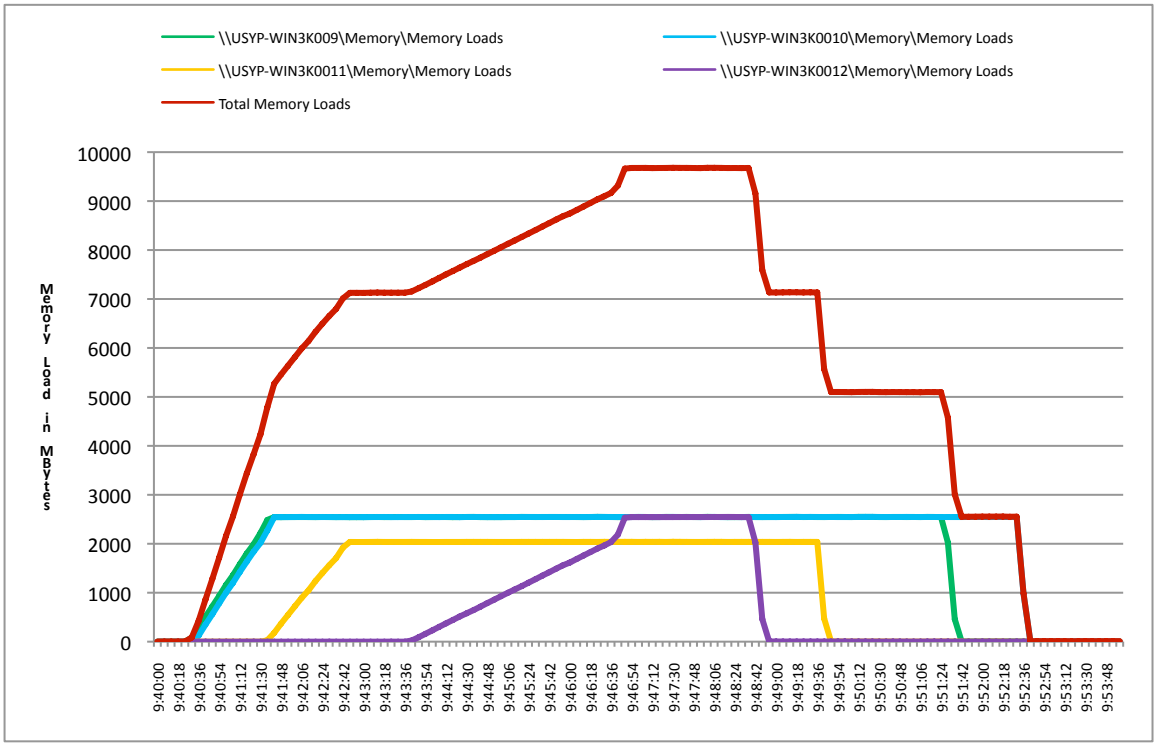


Figure 5. The Test Memory Workload Patterns

Memory Ballooning and Swapping Test Results

Once we start the test workload, we observe in Figure 6a using esxtop that ESX started the memory reclamation process via memory ballooning on the *usyp-win3k003* VM first. From the *esxtop* screenshot shown in Figure 6a, the **MCTLSZ** statistics of *usyp-win3k003* VM shows that 1.81 MB of physical memory has been reclaimed by the balloon driver and 83.71 MB of physical memory is going to be reclaimed. Furthermore, we see ballooning inflated for two of the VMs up to 2661 MB or 65% of the 4GB configured memory. Note the default percentage of configured memory that can be reclaimed from each VM by the balloon driver is 65% and this amount is indicated by the **MCTLMAX** statistics.

```
 9:43:03am up 12 days 18:50, 173 worlds; MEM overcommit avg: 0.59, 0.59, 0.59
PMEM  /MB: 32758   total:    800     cos,    677 vmk,   28346 other,   2934 free
VMKMEM/MB: 31533 managed:   1892 minfree,   3455 rsvd,   27627 ursvd,  high state
COSMEM/MB:    98    free:   1600  swap_t,   1600 swap_f:    0.00 r/s,    0.00 w/s
NUMA  /MB: 15411 (  179), 16319 ( 2754)
PSHARE/MB: 21312  shared,      1  common: 21311 saving
SWAP  /MB:     0    curr,      0  target:                  0.00 r/s,    0.00 w/s
MEMCTL/MB:     1    curr,     83  target, 31942 max
```

GID	NAME	MEMSZ	GRANT	SZTGT	MCTL?	MCTLSZ	MCTLTGT	MCTLMAX	SWCUR
321	usyp-win3k001	4096.00	4096.00	1523.54	Y	0.00	0.00	2661.89	0.00
322	usyp-win3k002	4096.00	4095.97	1559.95	Y	0.00	0.00	2661.89	0.00
323	usyp-win3k003	4096.00	4094.16	1933.47	Y	1.81	83.71	2661.89	0.00
324	usyp-win3k004	4096.00	4096.00	1709.11	Y	0.00	0.00	2661.89	0.00
325	usyp-win3k005	4096.00	4096.00	2156.06	Y	0.00	0.00	2661.89	0.00
326	usyp-win3k006	4096.00	4096.00	2251.42	Y	0.00	0.00	2661.89	0.00
327	usyp-win3k007	4096.00	4096.00	2349.46	Y	0.00	0.00	2661.89	0.00
328	usyp-win3k008	4096.00	4096.00	2449.04	Y	0.00	0.00	2661.89	0.00
329	usyp-win3k009	4096.00	4096.00	4127.51	Y	0.00	0.00	2661.89	0.00
330	usyp-win3k0010	4096.00	4096.00	4139.54	Y	0.00	0.00	2661.89	0.00
331	usyp-win3k0011	4096.00	4096.00	3597.45	Y	0.00	0.00	2661.89	0.00
332	usyp-win3k0012	4096.00	4095.95	1339.46	Y	0.00	0.00	2661.89	0.00

Figure 6a. Memory Balloon Statistics

As noted earlier, the Memory Ballooning process takes time and may not reclaim memory fast enough for ESX, forcing ESX to resort to VMkernel Swapping (as in vSphere 4.0) to actively reallocate memory during crunch times. Figure 6b demonstrates that- ESX reclaimed memory via memory ballooning and VMkernel swapping on all 3 VMs: *usyp-win3k001* to *usyp-win3k003*. We also notice from Figure 6b, excluding *usyp-win3k001*, that both *usyp-win3k002* and *usyp-win3k003* have up to the maximum of 2661 MB of balloon memory. This is the result of provisioning the first 3 VMs with no memory activities at the start of our test, making those VMs the primary candidates for ESX to use for memory reclamation. Since their idle memory was much higher than the other nine VMs, ESX chose them for memory reclamation ahead of the other nine VMs. In fact, our test shows that none of the other nine VMs have memory being reclaimed by the ESX (shown in Figure 6b), demonstrating how idle memory plays an important role in memory reclamation.

```
 9:54:42am up 12 days 19:02, 173 worlds; MEM overcommit avg: 0.59, 0.59, 0.59
PMEM  /MB: 32758   total:    800     cos,    677 vmk,   28351 other,   2929 free
VMKMEM/MB: 31533 managed:   1892 minfree,   3455 rsvd,   27627 ursvd,  high state
COSMEM/MB:    92    free:   1600  swap_t,   1600 swap_f:    0.00 r/s,    0.00 w/s
NUMA  /MB: 15411 ( 1695), 16319 ( 1233)
PSHARE/MB: 12768  shared,      9  common: 12759 saving
SWAP  /MB:  1103    curr,   1024  target:                  0.00 r/s,    0.00 w/s
MEMCTL/MB:  7563    curr,   7553  target, 31942 max
```

GID	NAME	MEMSZ	GRANT	SZTGT	MCTL?	MCTLSZ	MCTLTGT	MCTLMAX	SWCUR	SWTGT	SWR/s	SWW/s
321	usyp-win3k001	4096.00	1855.99	1282.12	Y	2240.01	2230.20	2661.89	0.00	0.00	0.00	0.00
322	usyp-win3k002	4096.00	851.03	238.65	Y	2661.89	2661.89	2661.89	583.05	561.36	0.00	0.00
323	usyp-win3k003	4096.00	913.95	621.99	Y	2661.89	2661.89	2661.89	520.13	463.59	0.00	0.00
324	usyp-win3k004	4096.00	4096.00	1715.04	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
325	usyp-win3k005	4096.00	4096.00	2157.55	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
326	usyp-win3k006	4096.00	4096.00	2253.39	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
327	usyp-win3k007	4096.00	4096.00	2364.88	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
328	usyp-win3k008	4096.00	4096.00	2460.01	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
329	usyp-win3k009	4096.00	4096.00	4136.78	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
330	usyp-win3k0010	4096.00	4096.00	4150.28	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
331	usyp-win3k0011	4096.00	4096.00	3602.80	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00
332	usyp-win3k0012	4096.00	4095.95	4152.49	Y	0.00	0.00	2661.89	0.00	0.00	0.00	0.00

Figure 6b. Memory Ballooning and VMkernel Swapping Activities

In order to analyze the impact of memory ballooning and VMkernel swapping on server performance, we need to examine VM memory being reduced due to memory ballooning and the

magnitude and duration of memory paging activities by the VMs' OS due to insufficient physical memory.

Based upon esxtop memory counter logs for a period of around 20 minutes, Figure 7 shows overall activities for ESX's free memory, the system wide memory balloon size and the reclaimed memory via Ballooning for each VM:

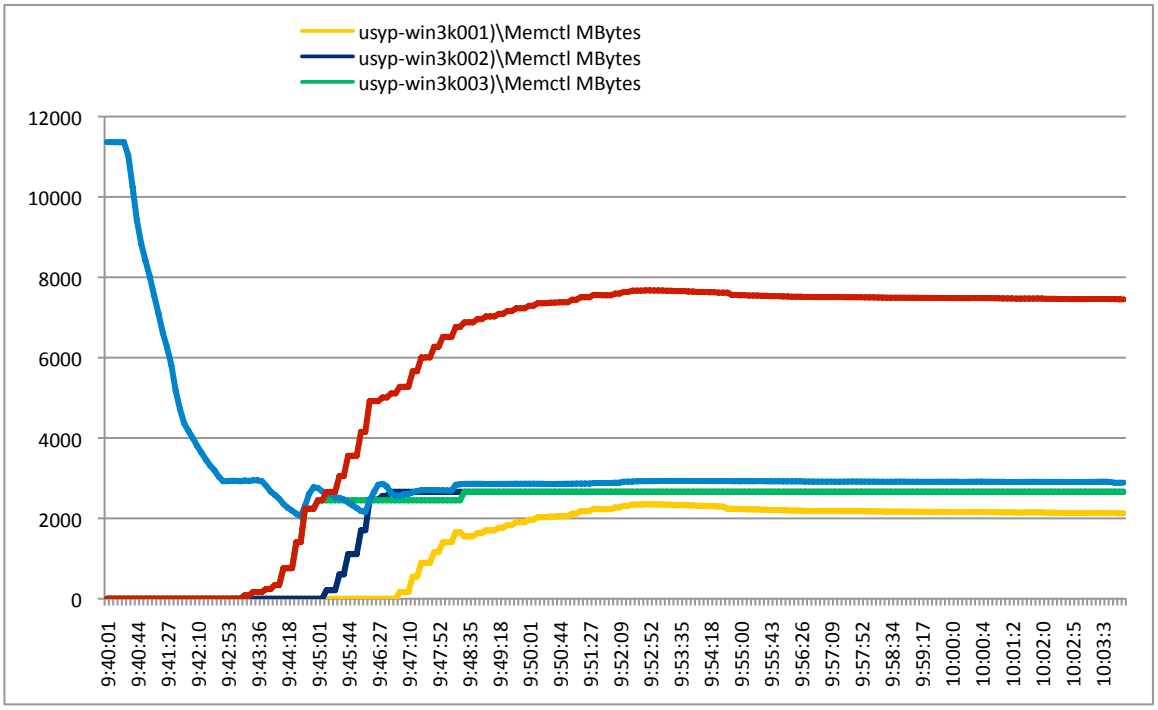


Figure 7. Memory Balloon Activities

ESX starts memory ballooning to reclaim memory from the *usyp-win3k003* VM when free memory drops to around 2.7GB.

The memory balloon activity shown in Figure 7 is consistent with what we have seen in Figure 6b. In other words, ESX reclaimed memory via memory ballooning and VMkernel swapping only from three VMs: *usyp-win3k001* to *usyp-win3k003*. We also noted that the balloon memory of all VMs stayed inflated long after the memory pressure was gone. We will revisit this scenario and discuss the conditions under which the balloon memory will deflate in a later section.

The vmware-tools Windows Perfmon log counters

The esxtop memory counter logs only provide performance information related to the ESX virtualization layer which is external to the VM. The memory counter logs which we collected via Windows Perfmon include (if the vmware-tools driver has been installed) both the VM internal counters as well as those related to the virtualization layer, e.g. "\\host\VM Memory\Memory Ballooned in MB". This dual-view monitoring allows to us to identify the internal performance bottlenecks that may be caused by memory ballooning and/or VMkernel swapping.

Figure 8 shows available memory and its internal paging to disk activities as seen by the *usyp-win3k003* Windows OS:

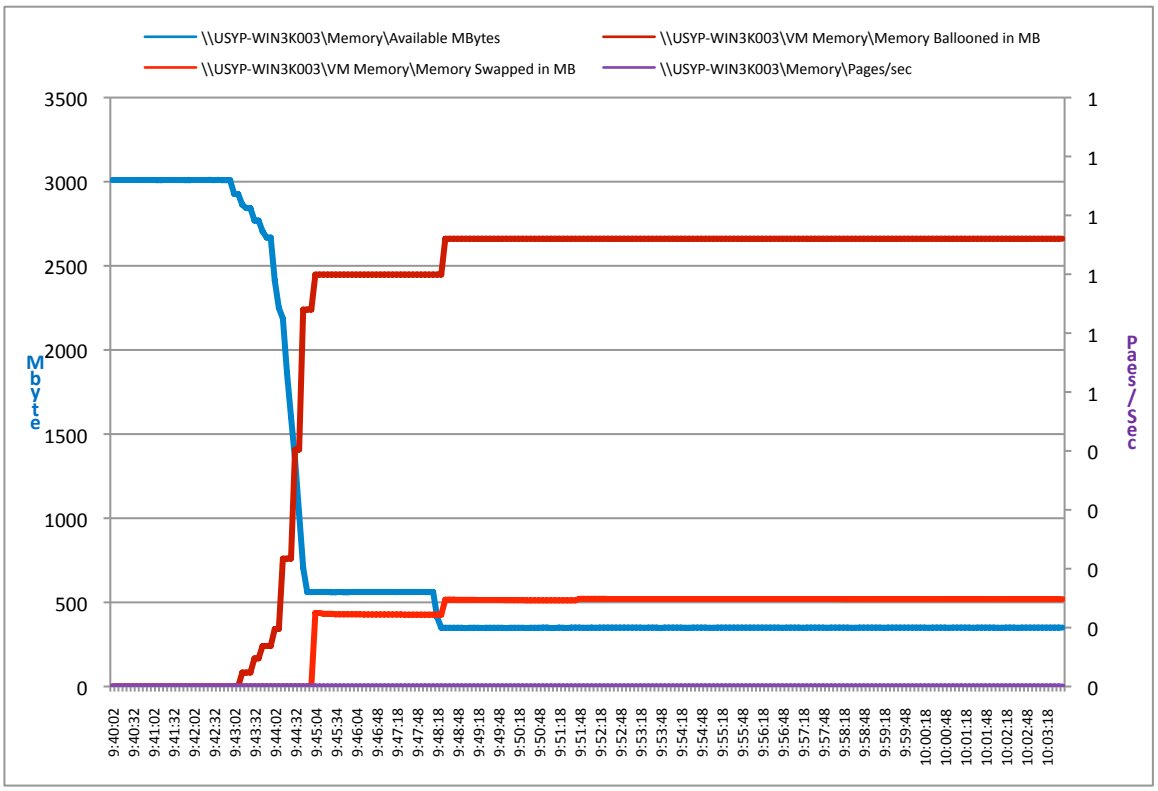


Figure 8. ***usyp-win3k003*** **VM Memory Activities**

Even though memory ballooning may be seen as undesirable, depending on the type of application, Figure 8 shows that the VM's performance may not suffer since there was little or no paging activity resulting from the balloon's inflation to reduce the VM memory. The question now is whether the VM's performance has been affected by VMkernel swapping. We will examine the potential performance penalties of VMkernel swapping in the next section.

More about VMkernel Swapping and the VM Swap File

For each VM, ESX sets up a single swap file for use in the event of memory shortage. The VM swap file is not created until the VM is powered on, and it is deleted when the VM is powered down. The VM swap file can be identified by its file extension of *vswp* and resides in the same disk folder where all the other VM files are located, by default. The VM swap file will not be used unless ESX needs to swap the VM's memory pages.

ESX normally creates a VM swap file with the same size as the VM's configured memory size, if there is no memory reservation configured. Since the VM swap file resides on the same disk storage as all the VM disks, we must consider potential disk usage when provisioning the storage for VMs, especially when Thin Disk Provisioning is configured. Thin Disk Provisioning is a new feature in VMware vSphere 4 that overcommits disk storage so that VM disk storage is allocated on demand, similar to the way dynamic memory allocation works on the ESX system.

IT administrators may not pay much attention to the VM swap file or even be aware of its existence, since the swap file is handled by ESX behind the scenes. With Thin Disk Provisioning, over-provisioning disk storage could lead to a situation where a VM can't be powered on due to

lack of disk space for the swap file to be created. With servers now accommodating up to 1TB of storage, enabling VM with up to 255GB of memory, Thin Disk Provisioning must be carefully planned out.

The general consensus is that VMkernel swapping is undesirable due to its high latency and should be avoided if possible. In the following test, we will show how swapping affects system performance.

Performance Monitoring for Memory Swapping

The first performance metric we will look at is the **%SWPWT** or **%Swap Wait** statistic (collected from the esxtop utility in batch mode) which indicates the percentage of time that the VM is waiting for ESX to swap memory. High **%SWPWT** values indicate that a VM's CPU time was mostly spent waiting for the ESX VMKernel to swap memory, definitely indicating performance issues. Other performance counters of interest include **SWR/s** (Swap Read Mbytes/sec), **SWW/s** (Swap Write Mbytes/sec) and **VM internal Pages/sec.** When greater than zero, all these counters indicate some form of memory overcommitment issue that needs further analysis.

Figure 9 shows the swapping activities on *usyp-win3k002* and *usyp-win3k003* as related to **Swapped Memory** and **% Swap Wait** time.

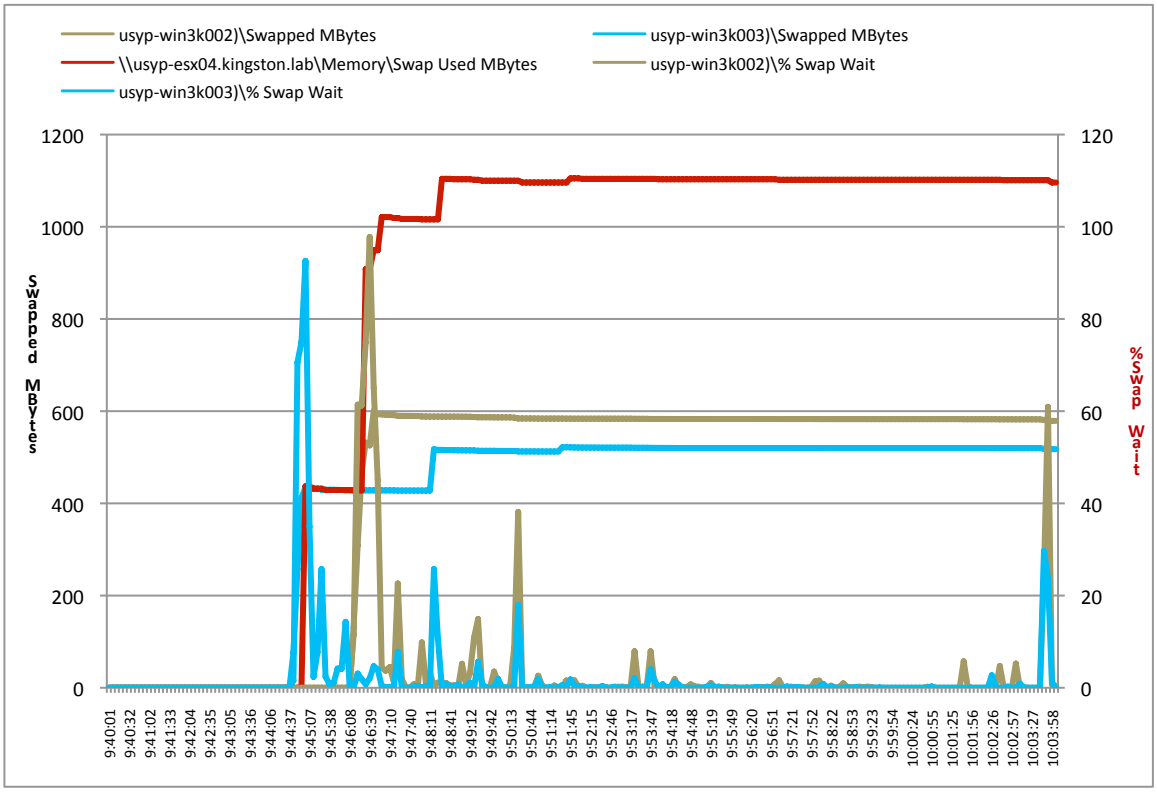


Figure 9. Performance as related to VMkernel Swapping

The **% Swap Wait** (as read on the right axis of the chart) indicates that there were times that the two VMs spent as high as 90 percent of their CPU time waiting for VMkernel swapping (during

which memory was swapped out to disk). When a VM spends most of its CPU time waiting for swap memory, its performance is likely to be severely impacted.

The Yin & Yang of Memory Reservations

Without proper planning and monitoring, there could be negative performance impacts resulting from excessive memory overcommitment. Looking at Figure 7 as the starting point, we need to ask: What if these three VMs are business-critical applications which must meet guaranteed service levels? To reduce the impacts of memory overcommitment, we must minimize VMkernel swapping as well as memory ballooning. Without increasing the amount of physical memory in the server, a memory reservation is the usual way to guarantee a VM will have the set memory capacity available for its use.

A VM administrator has the option to control the allocation of memory to certain business-critical VM by using memory reservation. In fact with VMware vSphere, we could configure a VM's memory resource allocation using the following 3 parameters:

- Memory Shares
- Memory Reservation
- Memory Limit

These parameters can be set at either the Resource Pool or the VM level. We will not cover Memory Shares in this White Paper. Figure 10 shows where we can configure these parameters at the VM level:

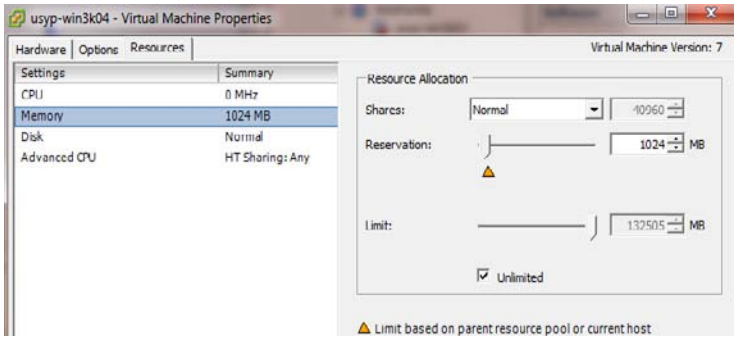
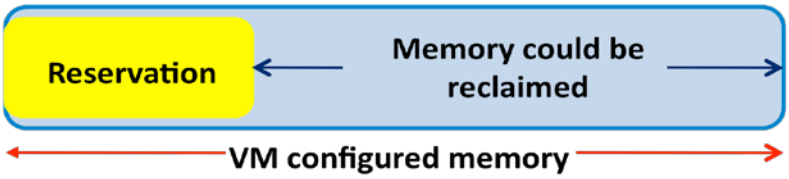


Figure 10. Individual VM Memory Resource Configuration Options

Memory Reservation specifies the guaranteed minimum amount of physical memory that will be allocated to the VM even during memory shortage. By default, this value is zero and ESX will have full control in allocating the VM memory up to its maximum configured memory. This can be illustrated by the following diagram:



Note that the VM memory will still be allocated on demand by ESX and starting from zero after power up, even if a virtual machine has a memory reservation. Furthermore, if a VM has not yet

accessed its full reservation, ESX can still allocate the unused memory to other virtual machines. *Only* after a virtual machine has accessed its full memory reservation, will the amount of memory reserved be retained. The reserved memory will not be reclaimed by the ESX, even if the VM becomes idle and stops accessing memory.

When a VM has a memory reservation, we note the following:

- The VM will have at least the amount of physical memory specified by the memory reservation value plus memory that will not be reclaimed by memory ballooning or VMkernel swapping.
- The maximum memory that could be reclaimed by memory ballooning is still the same: Configured Memory * 65%
- The size of the swap file that will be created by ESX when the VM powers up: (Configured Memory – *Memory Reservation*)

By setting a memory reservation, we can alleviate the potential performance impact of memory ballooning and/or VMkernel swapping by having the preset amount of memory always available to the VM. To show the memory allocation of ESX by using a *Memory Reservation*, we reran our previous memory load test, this time with 1.8GB of *Memory Reservation* configured on the *usyp-win3k003 VM*.

Figure 11 shows that *usyp-win3k003* has a 1.8GB Memory Reservation:

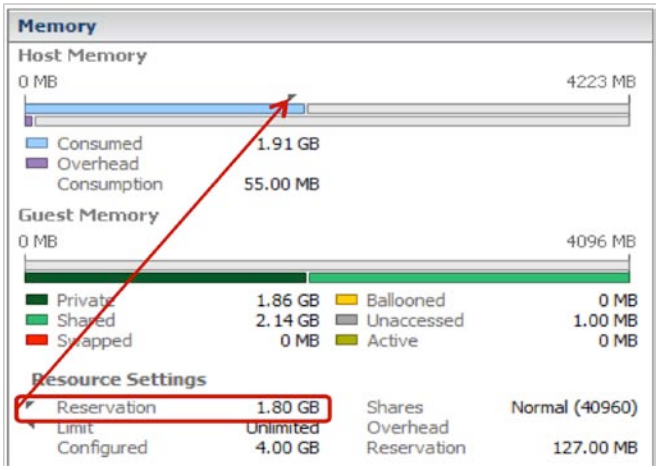


Figure 11. Memory Resource Allocation with Memory Reservation

After we configure *usyp-win3k003* with a memory reservation of 1.8GB, its swap file size is reduced by the same 1.8GB:

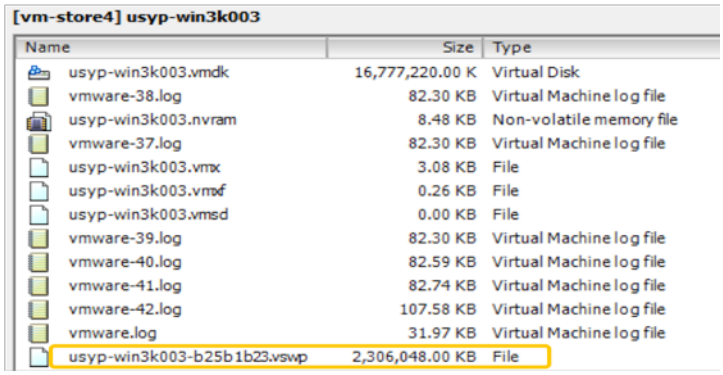


Figure 12. With Memory Reservation, the Swap File size is reduced

Figure 13 shows the memory activities and timing relationship between ESX free memory and Memory Ballooning. After running the same memory load test shown in Figure 5, and collecting memory log counters using the esxtop utility, we can see that, as expected, *usyp-win3k003* did not have any memory being reclaimed during the memory shortage period. However, the improvements in memory performance of one single VM (*usyp-win3k003)* using memory reservation had significant side effects, causing memory ballooning on the other 10 VMs as shown below:

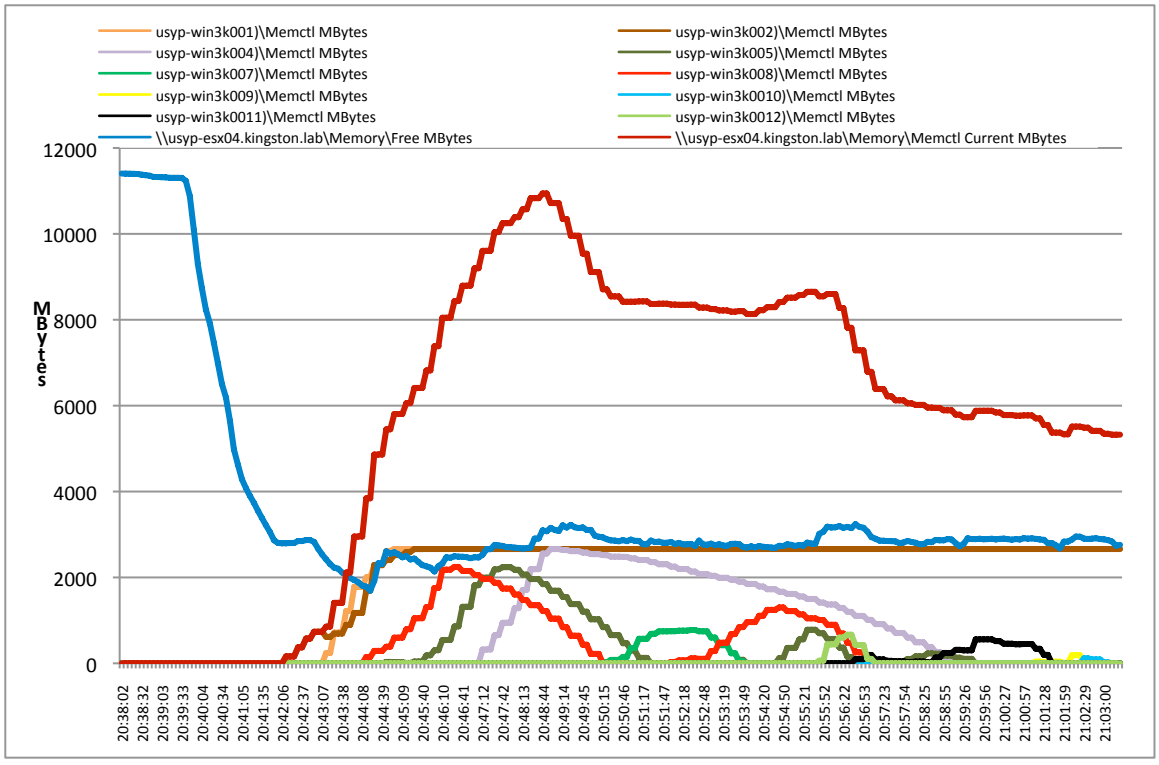


Figure 13. Memory Ballooning Activities

One interesting observation to note is that all memory balloons were later deflated except the two on *usyp-win3k001* and *usyp-win3k002;* this is due to the fact that ESX will not deflate the memory balloon(s) unless there are memory requests or memory activities from affected VM, even when ESX is no longer under memory pressure.

Now, let's look at the **Memory Swapped** and **%Swap Wait** in Figure 14. Now, a substantial percentage of *usyp-win3k004* CPU times is spent waiting for the VMKernel swapping as a result of setting the memory reservation on *usyp-win3k003*.

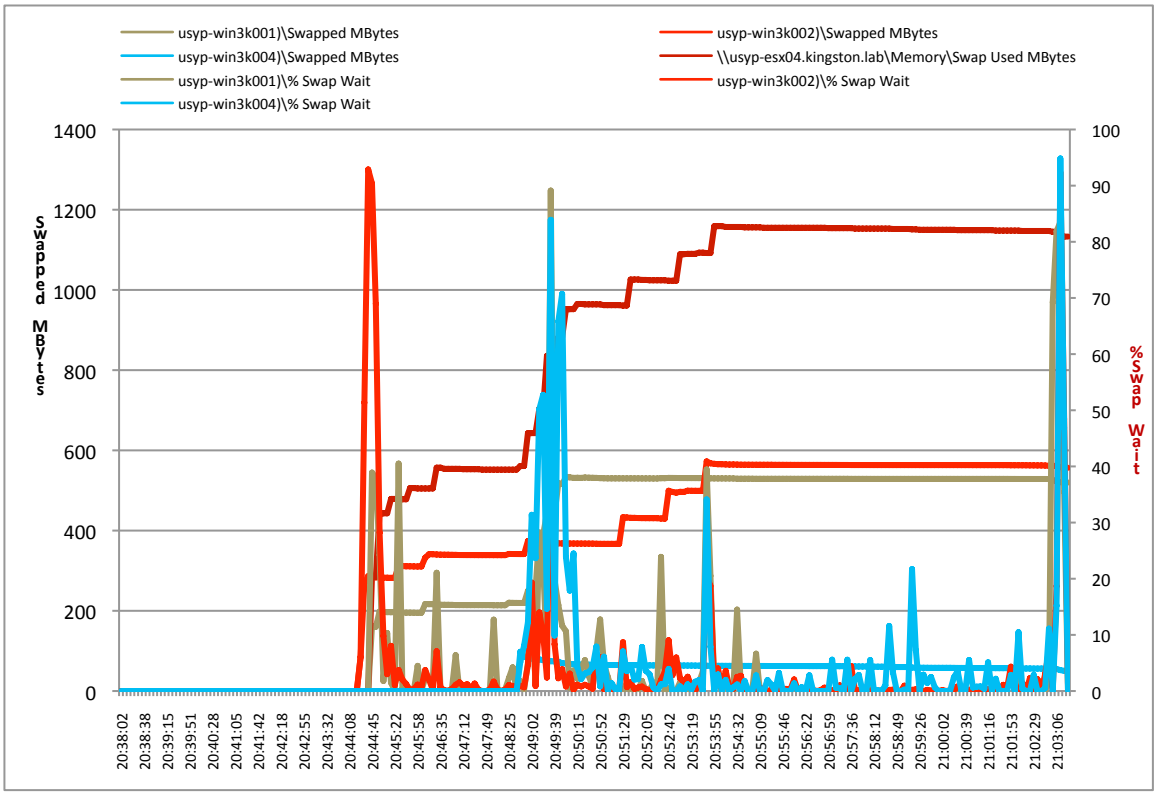


Figure 14. VMkernel Swapping Activities

To summarize: Even though we could improve the performance of specific VMs by setting memory reservation, our test results show that there may be unintended and adverse effects on the rest of the system. Furthermore, using memory reservation to improve performance did not solve the root cause of the system's performance issues: memory overcommitment.

Memory Limit – Solution or Problem?

The opposite of memory reservation is the memory limit. Just like a memory reservation, a memory limit is one of the parameters that can be set when configuring a VM memory resource at both the VM and/or the Resource Pool level. Memory limit has been one of the major causes of headaches for most Administrators facing unexplainable memory ballooning and/or VMkernel swapping activities — even in systems with no memory overcommitment.

Memory limit specifies the maximum physical memory that could be allocated to a particular VM. By default, the memory limit is set to the size of the configured memory. If memory limit is set lower than the configured memory, the VM memory layout is as follows:

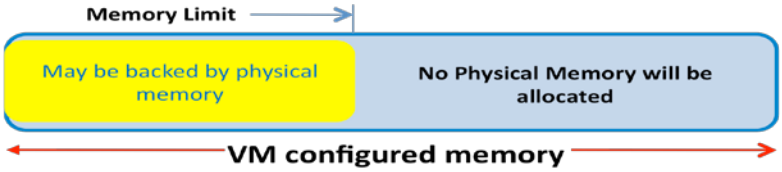


Figure 15. Memory Configuration with Memory Limit

An important insight here is to look at how ESX ensures that a VM with a memory limit will not consume more physical memory than its limit. If a VM tries to access more memory than its set limit, ESX will initiate memory reclamation by way of memory ballooning (within the VM itself) and/or VMkernel swapping.

From the VM's perspective, it has no notion about the memory limit being set and still assumes it has the entire configured memory that it could utilize. ESX must resort to memory ballooning and even VMkernel swapping to counter the increases in the VM's memory utilization that could potentially rise over the memory limit.

To demonstrate the memory limit impact on performance, we will set the memory limit of *usyp-win3k001* VM to 1024MB with a configured memory size of 4GB. In this case, we would expect ESX to initiate memory ballooning and VMkernel swapping to reclaim VM memory to counter the request for memory over the 1024MB memory limit. But since ESX can only inflate the memory balloon up to 65 percent of the configured memory or 2661MB in this case, the additional memory pressure must be relieved by swapping to disk!

With this unintended impact in mind, we tested how ESX enforces this memory limit by driving 2 GB of memory workload *on usyp-win3k001* for a test period of 3 minutes. To see where the ESX memory ballooning comes into play, we set up the workload to ramp up its full load in 15 seconds.

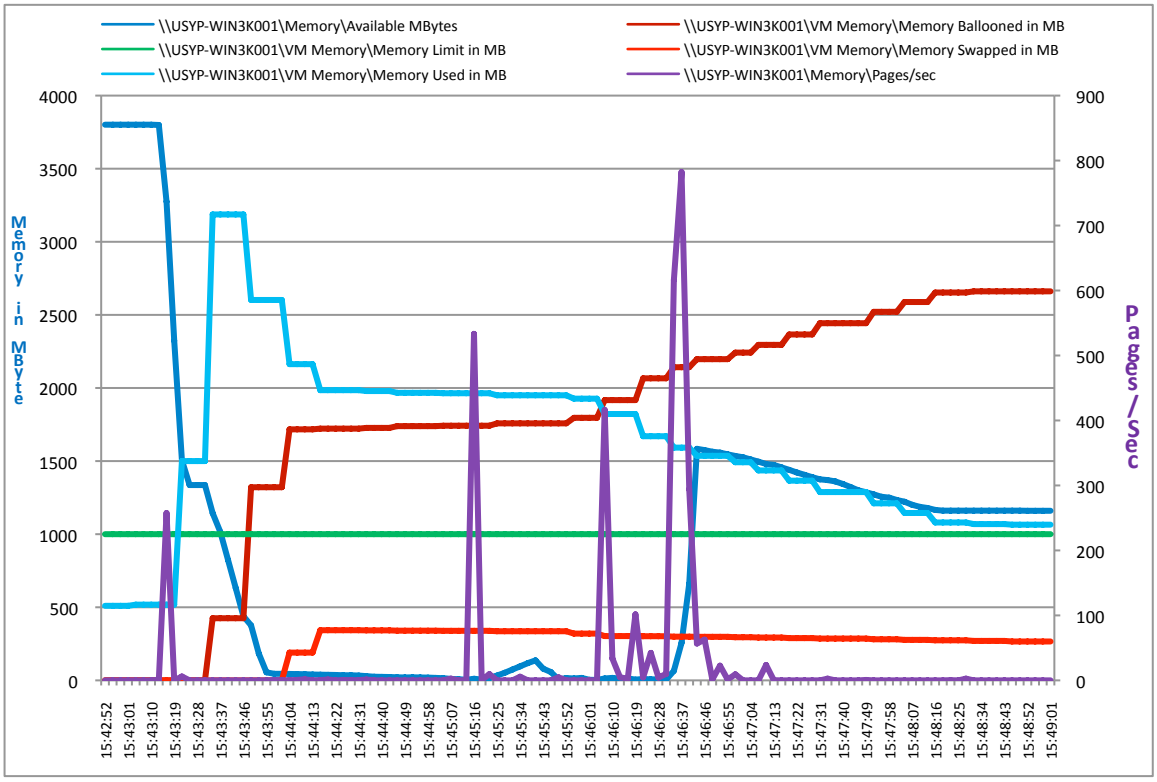


Figure 16. How Memory Limit is enforced through Memory Ballooning

Figure 16 shows that, as the Memory Used (shown in light blue) passes the 1024MB memory limit threshold, ESX starts memory ballooning. The memory balloon counters the increase of VM memory usage and forces the VM to relinquish free memory, eventually bringing down the VM memory (e.g., *"\VM Memory\Memory Used"*) to the level set by the memory limit.

Due to the excessive memory workload over the memory limit that we injected during the 3-minute period, the size of the memory balloon added to the increased memory workload which quickly depleted the available memory ("*\Memory\Available Mbytes*") shown in dark blue. We see extensive VM memory paging and VMKernel swapping. These results demonstrate why we should not use memory limit without first considering the usage case and understanding the potential performance penalties memory limit could cause.

Since the VM's OS has no awareness of any memory limit, the OS and all its applications may be impacted when a memory limit is exceeded, resulting in the ESX using memory ballooning and VMkernel swapping as counter-measures.

Summary

In this white paper, we demonstrated how ESX allocates memory based on VM memory activities, system workload and priority. When there is enough free memory or when ESX is in the ***high*** memory state, VM will always be allocated physical memory on demand. When memory is in shortage, ESX prefers memory ballooning over VMkernel swapping since swapping to disk imposes heavy performance penalties on the system. Because memory ballooning may take time and may not be sufficient to reclaim enough memory, our tests show that vSphere 4.0 starts VMkernel swapping to reclaim memory instead. Our tests show that memory ballooning may not be necessarily harmful to system performance but it does indicate that the ESX server has memory overcommitment issues requiring attention.

To explore memory overcommitment issues resulting from memory ballooning and VMkernel swapping, we ran several memory load tests on 12 identically-configured Windows 2003 server VMs with an average of 65% memory utilization running on a HP ProLiant DL 360 G6 system loaded with 32GB of memory. In order to exercise the system under various memory load conditions to see how ESX dynamic memory reallocation works, we utilized our own internally developed memory load generator utilities combined with sets of VMware PowerCLI scripts to automate all the testing.

Our testing demonstrated that using a memory reservation or memory limit could improve "localized" VM performance but did have negative impacts on the overall system performance while not addressing the root cause of the memory performance issues –excessive memory overcommitment. Thus, the use of memory reservation and memory limit for VMs should be carefully thought out so that peak VM workloads can be accommodated without unintentionally impacting individual VMs and even system-wide performance.

Understanding how ESX memory overcommitment works will aid administrators to better balance the Yin & Yang of configuring VM memory. We illustrated how performance could be affected by memory ballooning and worse, VMkernel swapping and/or VM Paging. Therefore, it is essential to not arbitrarily overcommit and size VM memory without first performing careful capacity planning and performance monitoring.

Thanks to:
Chethan Kumar and Fei Guo of VMware for their technical review, feature guidance and support

From Kingston:
Richard Kanadjian for his help in the majority of the technical review and editing.
Joe Maloney for his help in technical review and editing.
Edward Shen for his major contribution in developing the testing tools and testing scripts.