A White Paper

Guidelines for Overcommitting VMware Resources

HEROIX

Heroix
165 Bay State Drive, Braintree, MA 02184 USA
www.heroix.com, info@heroix.com

Guidelines for Overcommitting VMware Resources

Overcommitting resources in VMware means assigning more resources to Virtual Machines (VMs) than physically exist on the host running the VMs. VMware will allow you to overcommit, and manages host resources so that overcommitment works, but only up to a certain point. Once you've overcommitted too many resources, performance starts to degrade for both VMs and VMware hosts.

This paper will help you determine the extent of usable CPU and Memory overcommitment and guide you through distributing limited physical resources across multiple VMs so that you can get the most out of your host hardware. The topics we'll cover are:

1) Managing CPU and Memory Allocations
2) Managing Memory Allocations
3) Managing CPU Allocations
4) Conclusions

Heroix
165 Bay State Drive, Braintree, MA 02184 USA
www.heroix.com, info@heroix.com

Managing CPU and Memory Allocations

When you create a VM in vSphere, the VM is allocated a specific number of processors and amount of RAM. This can be a misleading in that it appears that the resources allocated to the VM have been set aside for its exclusive use, which is not the case. The actual behavior is that the CPU and memory allocated for the VM are upper limits, and the hypervisor provides resources to the VM as they are needed.

For example, if a VM is assigned 4 GB RAM, but the memory usage on the Guest OS is never > 50%, then the VM will only use 2 GB of the host's RAM, plus a small amount for management overhead. If a program running on the Guest OS needs more memory, it will get the additional memory it needs up to the maximum 4 GB.

One of the consequences of providing resources as needed rather than as allocated is that this makes overallocation possible. As long as the resources actively in use across all the VMs at any one time are less than the total resource capacity on the host, the hypervisor will be able to distribute resources between the VMs and transparent to the Guest OS's. But - if you push overallocation past that point where the amount of resources actively in use on the VMs exceeds the total amount of resources on the host, then performance will be severely degraded. A Guest OS with 4 GB RAM will not be able to access memory it thinks it should have, and processes or the Guest OS can crash.

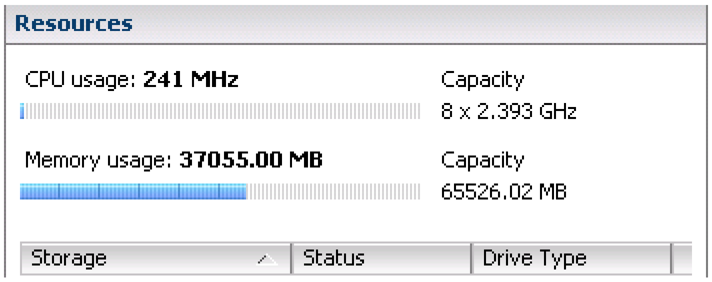
This begs the question: if overallocation can cause problems, then why bother? The answer is that not all VMs are created equal. For example, Virtual desktop VMs will underutilize resources most of the time and if you're facing pressure to provide as many desktops as possible, then overallocation is the most efficient use of limited resources. But if you're running a resource intensive VM like a SQL server - the problems caused by performance degradation would outweigh any benefit of maximizing resource usage.

That being said, how does VMware's hypervisor manage CPU and memory? And, how can you tell when you're reaching the point where you've overallocated too much?

Memory management

The basic terms for memory in VMware are:

- **Host Memory**



Resources

CPU usage: **241 MHz** — Capacity 8 x 2.393 GHz

Memory usage: **37055.00 MB** — Capacity 65526.02 MB

vCenter with 37GB consumed host memory out of 64 GB physical capacity

  - **Capacity**
    The physical memory available on the host.

  - **Consumed Memory**
    Total memory in use on the ESX host, which includes memory used by all running VMs and VMware management overhead.

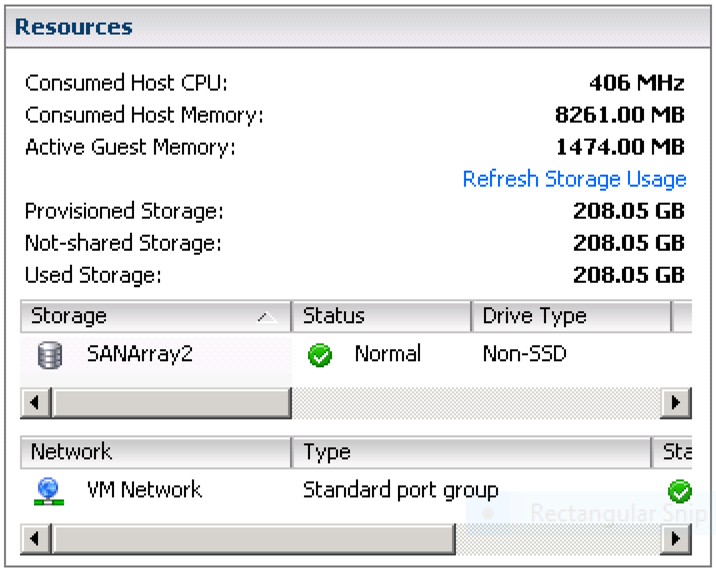
  - **mem.minFree**
    The minimum free memory threshold used to trigger the hypervisor to reclaim memory from VMs. The mem.minFree value is calculated as a percentage of memory capacity, with 899MB reserved for the first 28GB of host memory, and 1% of memory for every GB beyond 28GB:

Host Memory	mem.minFree
<= 28 GB	899 MB
>28 GB	899 + ((Memory Capacity) - 28 GB)* .01

For example, for a 64GB server:

mem.minFree = 899 M + (64GB – 28GB)*.01 = 899MB + 369MB = **1268 MB**

- **VM Memory**



Resources

Consumed Host CPU:	406 MHz
Consumed Host Memory:	8261.00 MB
Active Guest Memory:	1474.00 MB
	Refresh Storage Usage
Provisioned Storage:	208.05 GB
Not-shared Storage:	208.05 GB
Used Storage:	208.05 GB

Storage	Status	Drive Type
SANArray2	Normal	Non-SSD

Network	Type	Sta
VM Network	Standard port group	✓

vSphere display of a VM's Consumed Host and Active Guest Memory

- **Provisioned Memory**
  The amount of memory allocated to a VM plus the hypervisor overhead needed to manage the VM. As mentioned previously: VMs start with only the memory they require for startup, and add more as needed up to the amount of provisioned memory.

- **Consumed Memory**
  Current level of memory consumption for a specified VM.

- **Active Guest Memory**
  Estimate of memory actively in use by the VM's OS.

  If a VM requires more memory, the hypervisor grants memory up to its allocated amount. However, the VM does not signal to the hypervisor when it no longer needs memory and has unused capacity, and even if VMware Tools are installed, the hypervisor has no way of determining if there is unused memory on the VM. To gauge how much memory is actively used by a VM, the hypervisor checks a random sample of the VM's allocated memory and calculates the percent of the sample that is *actively* being accessed during the sampling period.

How does VMware manage memory?

1. VMs can be configured with a **memory reservation**, which is memory reserved for the exclusive use of the VM, and which must be available for the VM before it can be powered on. This memory is not subject to reclamation by the hypervisor if the host begins to run low on memory. The VM's memory allocation can be higher than its reserved memory, but the VM will always have at least its reserved memory.

2. If a VM does not have a **memory reservation**, when it is powered on it receives only the memory it needs from the hypervisor up to a maximum of the amount it has been allocated. The metric for this is **VM consumed memory**.

3. VMs receive additional memory as needed from the hypervisor, up to the VM's allocated amount. This is seen as an increase in the **VM's consumed memory** metric.

4. The memory for all the VMs and their management overhead is the **host's consumed memory**. The memory "state" on the host is based on how much free memory is available, ranging from High (preparing for memory reclamation) to Low (severe memory shortage, performance is degraded).

   The following chart displays the relationship between the mem.minFree threshold and the reclamation techniques that are triggered by a memory shortage. Note that as the host memory shortage increases in severity, the memory reclamation has increasingly severe effects on the VM's performance.

Host consumed memory vs. mem.minFree	**Memory State**	**Memory Reclamation Techniques**	**Potential Effect on Performance**
400%	High	Break down large memory pages	None
100%	Clear	Break down large memory pages + TPS	None
64%	Soft	TPS + Ballooning	Swapping on VM Guest OS
32%	Hard	TPS + Memory Compression + Swapping	Swapping on both VM Guest OS and host OS

16%	Low	Memory Compression + Swapping + Block VMs from allocating memory	Swapping on both VM Guest OS and host OS, VM Guest OS’s could crash

5. In the *High memory state*, VMware prepares for a possible memory shortage. The hypervisor will begin to break down large memory pages (2 MB) into smaller memory pages (4 KB) and check the smaller pages for duplicates.

6. In the *Clear state*, the hypervisor uses **Transparent Page Sharing** (TPS) to de-duplicate identical memory pages. Duplicate pages can be shared either within a VM or between VMs. From the perspective of the VM, TPS is “transparent”: the amount of memory the Guest OS sees is unchanged, but freeing up duplicate pages reduces the amount of memory used on the host.

   Note that in VMware 6.0 TPS is disabled by default between different VMs for security considerations.

7. If TPS does not recover enough memory, and the host enters the *Soft state*, the hypervisor uses ballooning to reclaim consumed memory that is not active. **Ballooning** works as follows:

   - The hypervisor contacts a balloon driver installed on the VM’s Guest OS as part of VMware Tools. If the VM does not have VMware tools installed, then ballooning won’t work.

   - The hypervisor tells the balloon driver to request memory for a balloon process on its Guest OS.

   - The Guest OS allocates memory to the balloon process. That memory is now unavailable for other processes on the guest OS.

   - From the Guest OS perspective, it still has all the memory it did before, but the portion used by the balloon process is no longer available. This can cause performance degradation for the VM if it needs to swap memory to keep other processes running.

   - The balloon driver contacts the hypervisor with the details of the memory that it has been allocated.

- The hypervisor removes the ballooned memory from the VM, lowering host memory consumed by that VM.

- If memory problems are resolved on the host, memory can be returned to the VMs by "deflating" the balloon memory and re-allocating it to the VM.

8. If memory reaches the *Hard state*, the hypervisor starts swapping and looks for memory pages to compress by at least 50%. At this point, performance is severely degraded not only for the VM Guest OS's, but for the host OS.

9. At a *Low memory state* the additional measure of blocking VM Guest OS's from accessing memory that has been allocated but not yet consumed is introduced, which could potentially cause Guest OS's to crash.

VMware Memory Metrics to Monitor

Data Collected for	**Metric**	**Threshold**
Host	Consumed Memory	4 * mem.minFree
VM (by hypervisor)	Active Guest Memory	Check vs OS Used Memory
Guest OS	Free Memory	OS Free Memory baseline
Guest OS	Paging	OS Paging Baseline

Recommendations for VM Memory

- Adjust allocated memory based on observed Guest OS memory metrics. For Guest OS's running memory intensive applications use observed Guest OS memory baselines in conjunction with the application vendor's memory recommendations.

- Check Guest OS memory use through memory reports that collect data from Guest OS performance metrics.

- Use memory reservations for Guest OS's running memory intensive applications (e.g. databases).

- Ballooning uses the hypervisor's Active Guest Memory estimate to determine how much memory can be reclaimed from a VM without unduly affecting Guest OS performance. If the Active Guest Memory estimate is significantly smaller than the VM's actual memory use, then ballooning could lead to paging on the Guest OS, degrading its performance. If Active Guest Memory is being underestimated for a VM, then either configure a memory reservation to protect the memory, or make sure that host memory is not overcommitted.

- Monitor Host Consumed memory on hosts where memory has been over allocated. If free memory is less than mem.minFree, memory reclamation can be observed as unexpected drops in Host Consumed as memory is reclaimed from the VMs, or as low free memory on the Guest OS as ballooning locks Guest OS resources.

- While ballooning is occurring, monitor free memory and paging on the VM Guest OS's. Move VMs to hosts with more memory before ballooning causes performance degradation on the VMs.

- If ballooning does not resolve low memory on the host, move or power down VMs before reaching the Hard memory state (32% of mem.minFree). Memory compression and swapping cause severe performance degradation.

CPU Management

The way the hypervisor distributes CPU is very different from the way it distributes memory. Host memory is a fixed quantity while CPU resources are queued, therefore over allocated CPU will cause increased wait times rather than shortages.

The concepts and terms used to discuss VMware CPU management are:

- **Socket**

  The socket is the connection on the host's motherboard for the CPU processor. VMware is licensed based on the number of sockets in use.

- **Cores - Physical CPUs (pCPU)**

  Each CPU processor on the system is made up of one or more cores. The core is the component that executes the instructions sent to the CPU and is referred to as a physical CPU, or pCPU in vSphere.

- **Hyperthreading - Virtual CPUs (vCPU)**

  CPU processing requests are queued and sent to the pCPU, and if the request at the front of the queue has to wait for resources, then the processor is idle while it's waiting. Hyperthreading adds a second instruction queue for a pCPU that can be scheduled in conjunction with the first queue to minimize wait times for both queues.

  A core that uses hyperthreading is considered to have 2 "logical" processors per each pCPU, although in practice the performance gain from hyperthreading is closer to 30% than it is to double. In VMware, logical processors are seen as "virtual" processors, or vCPU.

- **Processor Speed**

  How quickly a CPU can process requests is influenced by the processor's speed. Processor speed is usually measured in cycles/sec, or Hz, and current processors run at a multiple of GHz (or 1 billion cycles/sec).

  vSphere will display the cumulative CPU speed over all the processors, and will display usage for VMs in terms of the number of CPU clock cycles they have used.

- **NUMA (Non-Uniform Memory Architecture)**

  Before NUMA was developed, computers were designed so that each processor accessed all the system memory using the same system bus (i.e. system path). When multiple processors tried to access memory at the same time, there was contention between the CPUs that slowed down processing.

  NUMA addressed this contention problem by breaking the system up into nodes, where each node had one or more processors which are associated with a portion of memory. This allows CPUs to access memory on their local node over a local bus without having to contend with other CPUs. Basically, it's the difference between merging onto a highway onramp during a traffic jam versus pulling onto less busy side street.

Note that NUMA works best when the CPUs access local memory on their local NUMA node, but they can access memory on other nodes, albeit less quickly. Hypervisors will generally try to avoid “Wide VMs”, which occur when CPU and memory resources used by a VM are on multiple NUMA nodes.

However, if the number of processors allocated to a VM requires that it run as a wide VM, then the hypervisor will try to allocate vCPU and memory resources symmetrically. For example, if a NUMA node has 8 vCPUs, and you create a VM that is allocated 12 vCPUs, the hypervisor will split the CPU allocation across 2 NUMA nodes of 6 vCPU each.

- **Virtual NUMA (vNUMA)**

  Applications such as Microsoft SQL are NUMA aware, and can use NUMA’s localized memory to improve performance. If SQL is installed on physical hardware, it can detect that NUMA is available and take advantage of the architecture. However, because VMs are built based on virtualized hardware created by the hypervisor they do not have direct access to the underlying NUMA architecture.

  To allow NUMA aware applications to take advantage of NUMA while they’re running in VMs, VMware provides a virtualized NUMA (vNUMA) environment. This architecture assigns the VM processors and memory across multiple NUMA nodes and presents a virtualized NUMA architecture to the VM’s operating system. In our previous example of a wide VM, a SQL server using vNUMA would see 2 NUMA nodes, as opposed to seeing 12 processors without vNUMA.

- **Co-stop**

  VMware schedules all the vCPUs for a VM at the same time. If all the allocated vCPUs for a given VM are not available at the same time, then the VM will be in a state of "co-stop" until the host can co-schedule all vCPUs. In its simplest form co-stop indicates the amount time after the first vCPU is available until the remaining vCPUs are available for the VM to run.

- **CPU Ready**

  VM CPU Ready is a measure of the time a VM has to wait for CPU resources from the host. When overallocation of CPU resources becomes severe the CPU ready value will increase.

VMware CPU Metrics to Monitor

Data Collected for	**Metric**	**Threshold: alert if:**
Host	CPU utilization	> 80% (if > 90%, reaching overload condition)
VM	CPU Ready	> 5%
VM	Co-Stop	> 3%
Guest OS	CPU utilization	> 80% (if > 90%, reaching overload condition)

Recommendations for VM CPU

- vCPU overallocation up to 3 x VM CPU's per host CPU usually doesn't cause performance issues

  It does depend on the function of the VMs running on a host, but in general you can allocate 3 times more CPUs than exist on the host without seeing performance issues. However, higher ratios will begin to introduce performance issues.

- Don't allocate more vCPU's than you need.

  When creating a VM, start with the number of CPUs recommended for the Guest OS or for a specific application, and only increase the number of CPUs if the baseline value for Guest OS CPU utilization indicates that more CPU is needed. Remember, more vCPU's can also mean more co-stop time as the VM waits for all the allocated processors to be available, so more CPUs aren't necessarily better.

- Consider NUMA architecture when creating VMs

  NUMA information is not available through the vSphere client, so you will need to know your host's architecture. If NUMA is in use, keep the number of vCPU assigned to a VM to those within a NUMA node if possible.

Note that the hypervisor does not take hyperthreading into account when determining how many vCPUs are available in a node - it only counts physical processors (pCPU). The settings can be overridden using the advanced VM property *numa.vmcpu.preferHT* to True.

- Take advantage of vNUMA

  vNUMA is available as of Virtual Machine Version 8, and enabled by default for VMs with more than 8 vCPUs. If a VM needs more memory than is available within its vNUMA nodes but doesn't need more vCPU, you can use advanced VM configuration settings to adjust the number of vCPUs.

- VMs that communicate frequently can cause NUMA imbalance

  The hypervisor will position VMs that communicate frequently with each other on the same NUMA node to try to optimize I/O. However, this could lead to a load imbalance with too many VMs on one NUMA node, and not enough on another. This positioning can be disabled by changing the advanced Numa.LocalityWeightActionAffinity setting to 0.

- Move VMs to a different host if VM CPU Ready or host CPU utilization is too high

  If host CPU utilization is > 80%, there is the potential that CPU ready values for VMs will begin to increase. Plan to move VMs to different hosts before host CPU utilization reaches 90% or if CPU Ready values for VMs is > 5%.

- Move VMs to different hosts or reduce the number of allocated vCPUs if Co-Stop is high

  If co-stop is > 3%, then the VM is waiting for all its assigned vCPUs to be available. If you cannot reduce the number of vCPUs allocated to the VM, then move the VM to a host that has less competition for resources.

Conclusions

- Overcommitting memory can make the best use of your resources, but monitor the host's consumed memory and the effect of memory reclamation on the performance of your VM Guest OS's. If you have VMs running memory sensitive applications, make sure you allocate enough memory for them, protect them from memory reclamation, or, if necessary, run them on hosts which do not overallocate memory.

- Overcommitting CPU allows you to maximize use of host CPU resources, but make sure to monitor overcommitted host CPU use, CPU Ready and Co-stop percentages. Consider NUMA architecture and the effect of co-stop waits when creating VMs with multiple vCPUs..

About Heroix

Heroix has a 30+-year history of proven monitoring solutions, with products running on tens of thousands of critical servers. It offers fast, easy, affordable application and networking monitoring solution for physical and virtual environments. Download Longitude Now and you'll be monitoring and planning in just 10 minutes.

Heroix
165 Bay State Drive, Braintree, MA 02184 USA
www.heroix.com, info@heroix.com