VMware® Infrastructure 3

Advanced Technical Design Guide

~and~

Advanced Operations Guide

Two books in one!

Ron Oglesby
Scott Herold
Mike Laverick

Chapter 9: Resource Management

This chapter discusses the tools available to you for tweaking the performance of VMs. We can adjust resource parameters for CPU, Memory, and Disk on a per VM basis, or we can drop groups of VMs into "resource pools" to manage their CPU or memory. Resource pools merely allow us to treat VMs as groups, rather than individuals – and quickly apply resource settings to them – thus lowering your administrative burden.

It is possible to cap, or "limit," and also guarantee a minimum, or "reservation," of either CPU or memory to a VM or a resource pool. Alongside these unchanging and hard-coded limits or reservations, we can have a more dynamic control over VM's usage of CPU, memory or disk. We can use VMware's proportional "share" system which responds to changes in resource demand relative to each VM and ESX host. At an extreme level it is possible to peg a resource to a VM with such features as CPU affinities – hard-coding a VM to have exclusive access to a CPU.

Technically, I really should mention that VMware Distributed Resource Service (DRS) is an invaluable resource management tool. DRS integrates very closely with VMware High Availability (HA) so it was decided to show these two products together in the "High Availability" chapter of this book.

In this chapter I want to use an airline analogy to explain how resources are managed in ESX. Each plane will represent an ESX host, and a fleet of airplanes will represent a VMware DRS cluster. Every plane has fixed capacity in terms of the number of passengers it can comfortably accommodate. Each of the "seats" will represent a VM. If the airline has 200 seats per plane and ten planes that is a total of 2000 seats. But the 200 seats per plane do represent a fixed limit on capacity. Fundamentally, though these 2000 seats exist as a logical capacity, a passenger can only board one plane – just as a VM can only currently execute one server, not a cluster. Access to the plane is controlled by a unique system of reservations which is intended to guarantee not that you fly – but that a certain quality of travel like business class will be provided. In contrast, economy passengers do not make reservations. They merely turn up and hope there will be capacity to fly.

Setting Limits

As stated a moment ago it is possible to impose limits upon a VM or resource pool. This can be done for CPU resources (by MHz) and for memory (by MB). In fact, limits are imposed on VM from the perspective of memory when you first define a VM. When you create a VM you set the maximum amount of memory it can have during the "New Virtual Machine" wizard. If a VM demands more memory it cannot exceed this amount allocated to it – even if free physical memory exists. At first glance this seems quite restrictive, however, from an architecture point of view, it has to happen. If we have poorly written applications and operating systems we would want to avoid the situation where a VM was able to drain a physical host of its entire available RAM – thus causing a reboot of the physical server. If we use our analogy it's not possible for a single passenger on the plane to take up all the space at the expense of the other passengers. Additionally, there are some fixed limits we cannot exceed. If there are only 200 seats on the plane we cannot allow 201 passengers to board. Similarly, if the ESX host or pool resources are totally consumed – a limit has been reached. We cannot magically out of nowhere find additional resources - with one exception. When all memory has been depleted is possible for a VM to use its VMkernel swap file.

In contrast there are no default limits on CPU usage. If a VM demands CPU time, and that CPU time is available, then this is allocated to a VM. One possible reason to cap or limit a VM usage of CPU is that you know you have a poorly written application that regularly crashes. Perhaps when it crashes it "hogs" the CPU of the ESX host. Using CPU limits is one method (amongst many) to control these kinds of VMs.

Lastly, a word about terminology - in the ESX 2.x product the word "maximum" was used instead of the word limits. VMware changed the phraseology to "limits" as this is more meaningful and more clearly describes the feature. Additionally, VMware has moved away from allocating limits on CPUs by a percentage value and now prefers to use Mhz. MHz is a more accurate measure of CPU usage than percentages, which can be very misleading. In the context of CPUs, 10% of 1.44 MHz processor is decidedly different from 10% of 2.6 MHz processor.

Setting Reservations

In addition to limits, we can also use CPU or memory "reservations." One analogy that can help when thinking of reservations is to compare them to the plane reservations. These reservations are supposed to *guarantee* a resource in the form of a seat. Similarly, a CPU or memory reservation in MHz or MB is intended to *guarantee* the resource to the VM or resource pool. In this case there are no *default* reservations for CPU or memory, unlike the default limit on memory set when creating the VM. We can regard these reservations or guarantees as offering us a way of ensuring we meet certain performance levels. Perhaps you could even regard them as a way of meeting "service level agreements" (SLA).

Just like with a plane or hotel you must "meet" your reservation in order to board or check into a hotel. So, if you say that the memory reservation on VM is 256MB of RAM – and that amount of RAM is physically not available – you will be unable to power on the VM. VMware refers to this as "Admission Control." Similarly, if we configured a situation where a VM must get 1000 MHz of CPU time, but the physical host can only offer 500 MHz of CPU time the VM would not power on.

Using our airplane analogy, if a passenger makes a reservation for 10 seats in business class those seats must be there. Our airline imposes a very *special* definition of customer care that states if a business class reservation cannot be met, rather than pushing our customer in economy class, we refuse admission to the flight. Here we can see the weakness of all analogies; they don't work perfectly in all circumstances. Nonetheless, I want to persist with this analogy as it is helpful in most cases.

As with CPUs in ESX 2.x, VMware used to call "reservations" minimums. Again, the label change was introduced to be more meaningful.

Putting the concerns to one side for a moment there is very interesting and useful relationship between memory limits and reservations and the VMKernel swap file. The difference between the reservation subtracted from the limit determines the size of the VMkernel swap file. The usage of the VMkernel Swap file was mentioned in the previous chapter as an indicator of potential performance problems – but here I wish to delve a little deeper into different ways of

using it. Explaining how to use the VMKernel swap file is perhaps best done with a couple of examples.

Example 1: Difference between Limit and Reservation

I had a VM with a 512MB limit and 256MB reservation – powering on the VM would create a 256MB VMkernel swap file (512-256) and guarantee that the VM would receive 256MB of RAM.

Example 2: No difference between limit and reservation

If I set the limit to 512MB and the reservation also as 512MB and powered on the VM, ESX would not create a VMkernel swap file at all. It would run the VM entirely in a memory reservation of 512MB.

Example 3: Big difference between limit and reservation

If, on the other hand, the VM was given a 16GB limit, and the default of 0MB was used for the reservation a 16GB VMkernel swap file would be created.

With example 1, if I had an ESX host with 2GB of physical RAM I could run at least 8 VMs before running out of memory (2048MB/256MB). I would not be able to power on a 9th VM because there would be insufficient memory to meet the reservation guarantee. If all the VMs simultaneously wished to use memory up to the limits (512MB*8) I would get swap activity. What I hope is that this would be such an unlikely event that it would be "safe" to configure the system this way.

Perhaps you find this "memory over-commitment" a bit scary. You are concerned about the negative aspects of swap activity, and you wish to have a cast-iron guarantee that your VMs will *always* run in memory. If this was the case you could use example 2. By setting the VM's limit and reservation to be the *same* value no swap file is created – and the VM is guaranteed to always run in memory. However, on a 2GB system the effect of this policy would be very significant. I would only be able to run 4 VMs not 8 (2048MB/512MB). If I tried to create a 5th VM and power it on – it wouldn't, as all my memory would have been reserved for use by my other VMs.

I could imagine example 1 being configured by someone who is optimistic and is looking for very high VM to ESX host ratios or someone trying to run as many VMs with as few resources as possible. Alternatively, we could see example 2 as someone who is perhaps pessimistic or conservative, or someone who has so many resources there is no need to use the VMkernel swap file.

The last example, example number 3, is a warning. If you set extremely high values on memory, with no reservations the ESX host will generate an extremely large swap file. Just like with memory reservations, you need the physical MB disk space to create the VMkernel swap file. In example 3, if you didn't have 16GB of free disk space in the LUN where the VM is stored, it would not power on as there would be insufficient resources to guarantee the *difference* between the limit and reservation.

The Share System

Another system is at play in the gap between limits and reservations, called the "Proportional Share" system. Shares allow you indicate that when a resource is scarce that one VM or resource pool is more important than another. To use our analogy, it's like the airline treating a Hollywood star more importantly than the average guy on the street. Share values can be applied on per-VM basis or on resource pools. Unlike limits or reservations, which are fixed and unchanging, shares react dynamically to resource demands. The share value can be specified by a number (usually in multiples of 1,000) or by user-friendly text value of normal, high, and low. The important thing to remember about the share value is that it is only relevant when the resource is scarce and contention is occurring. If the resource is plentiful or VMs do not have to compete over resources the share value does nothing at all.

In my discussions with VMware I've been told that many customers do not use the proportional share system as much as VMware might like. Why might this be? Firstly, because customers frequently don't understand how shares work, and secondly, because shares only take effect when things are performing badly – a great many people try to configure their ESX hosts and VMs so this never happens. Personally, I am a big fan of the shares system. What especially appeals to me is its dynamic nature, its ability to react to changes.

If you or your customer is still struggling with the concept of shares you might like to try a couple of other analogies. You could see the share value like shares in a company that are quoted on the global stock exchanges. The amount of shares a company chooses to issue is up to it – what matters is the number or portion of your shares. The greater amount of shares you hold in a company the more of its resources you own. Therefore, a big share owner of 5000 shares has much more influence than a share holder with just 1000 shares. Perhaps you own as much as ¼ of the company, or 25% of its shares.

Here's another analogy I use regularly in courses. Imagine you have 3 children – one is a baby, the next a 5 year-old and the last a teenager. When you come home after a hard day's work they all demand your time. Here your time is like the CPU, and each of your children are pesky VMs making demands on your tired brain. Being a particularly cruel parent you decide to take a permanent marker – write 3000 on the baby's forehead, 2000 on the toddler's forehead, and lastly 1000 on the teenager's forehead. You decide you're like one of the ESX hosts you manage at work, and this is your parental strategy from now on!

In this scenario, when you are faced with contention (say when you come home from work) you would give the baby ½ (3000/6000), the toddler $^1/_3$ (2000/6000) and your teenager just $^1/_6$ (1000/6000) of your valuable time. Now when it's mid-evening you decide that the baby is tired enough to go to sleep. You're in luck tonight as he's out for the count in seconds – you now can give $^2/_3$ of your time to the toddler (2000/3000) and $^1/_3$ (1000/3000) of your time to the teenager. When the toddler goes to bed (after much crying and wailing normally) you are facing no contention at all. Just as you're settling down to watch your favorite sit-com the teenager comes down from her bedroom – perhaps the internet connection has failed or her games console has broken. She now decides this will be an opportune time to discuss why college is a waste of time and how she should really follow her favorite drug-taking band around the country. Now you can give all of your time to teenager – 1000/1000 – in persuading her that while a life of drunken debauchery might have its appeal, it won't lead her to a prosperous career in IT like yours. Finally, everyone goes to bed – contention is over and you get the opportunity to get some well-earned z's. But then at 3 a.m. a sound is heard from the baby's room which grows into crying. You're out of luck, and it is your turn and not your partner's to feed the baby. However, rest assured as long as you get to the baby quickly, it will not wake the others – and you are able to give 100% of your time to getting back in bed as quickly as possible!

All joking aside, the analogy does illustrate some points. Firstly, that share value adjusts depending on the level of contention. Secondly, that when there is no contention the share value does nothing at all. Thirdly, that when you become a parent you will have no time to yourself whatsoever!

Lastly, you should know that there is another way of setting the share value which is by using friendly labels of "High, Normal, and Low." These offer novices a more intuitive way of dividing up resources. You see these whenever you create a new VM. You can use these text labels on a VM and also in resource pools. If you are going to use them you should know what actual settings apply.

- **High**

  Allocates 2000 shares per virtual CPU:

  20 shares for every 1MB allocated to the VM

- **Normal**

  Allocates 1000 shares per virtual CPU:

  10 shares for every 1MB allocated to the VM

- **Low**

  Allocates 500 shares per virtual CPU:

  5 shares for 1MB allocated to the VM

As you can see, high is twice as much as normal and four times as much as low. There is also another assumption at play here. VMware assumes that the more memory you assign to a VM the more sensitive it is to a lack of memory. So when contention takes place the VM "wins" a greater slice of memory resources. This assumption might not always be the case (although it frequently is). You could have a memory intensive application that is not business critical.

CPU Affinities

One extreme method of controlling the VM's access to CPU resources is to "peg" it to a specified CPU. As was mentioned in the previous chapter, internally to physical server the VMkernel dynamically moves the VM across to work on the best CPU inside the ESX host. We can switch off this feature using CPU

affinities on the properties of a VM. I would regard this configuration as a last resort. Firstly, configuring it is very administration intensive. Not only do you have to configure the VM in question to use only CPU3 for example, you also have to configure every other VM *not* to use CPU3 – to truly dedicate a VM to given CPU. Secondly, CPU affinities are incompatible with VMotion. Thirdly, as DRS is effectively an automated VMotion for performance – CPU affinities also break DRS. Removing CPU affinities on a VM running on a ESX host which is already a member of DRS cluster is possible, but very convoluted. Therefore, I consider CPU affinities a very last resort.

Resource Pools or VM Settings

There are two main ways to apply many of these settings – limits, reservations, and share values. Resource pools only affect CPU and memory resources – so if your goal is to control disk and network activity, these must be done on the properties of the VM or a vSwitch, respectively. Despite this limitation, CPU and memory resources are very critical, and resource pools offer a much more effective way of applying limits, reservations, and share values. By its nature, right-clicking each VM and setting these values is very administration intensive, whereas dragging and dropping a VM to the correct VM is an easy task. If you are trying to calculate the total share value it is easier to compare a small number of resource pools, rather than comparing a large number of VMs.

Resource pools can be live in two main places – hanging off a stand-alone ESX server which divides up the resources of a single host into smaller units or pools, or alternatively, they can be created on VMware Cluster which divides the total resources of many servers into pools. Once you have the concept of resource pools and the way they function (they are the same if they are on a stand-alone or a cluster, but they really come into play in clusters where the need to divide resources up logically might more pressing) you could create resource pools based by department (sales, accounts or distribution), function (web servers, database, or file servers), or by IT Infrastructure (test, development, production).

If we apply our analogy to this we can see each of the airplanes as representing an ESX host. A DRS cluster represents the collective capacity of all my airplanes, but fundamentally a passenger can only fly on one airplane – not on two simultaneously. Our air-traffic controllers are very clever guys who can detect

some airplanes stretched to capacity and move passengers from one plane to another while they are flying. This is called VMotion in ESX.

Putting It All Together: Creating Resource Pools

In this example I am going to run two VMs with a CPU intensive process. Initially, there will be no contention. I will introduce contention by forcing them to run on the same physical CPU. Then I will create resource pools with different share values to show both how resource pools help you control resources and demonstrate that the shares feature does work dynamically.

Creating CPU Intensive Events

1. On the *same* ESX host power on two Windows VMs.
2. On the root C: create two VBS files called cpubusy.vbs.
3. Cut & Paste the following code to the cpubusy.vbs script:

```
Dim goal
Dim before
Dim x
Dim y
Dim i

goal = 2181818
Do While True
      before = Timer
      For i = 0 to goal
            x = 0.000001
            y = sin(x)
            y = y + 0.00001
      Next  y = y + 0.01
WScript.Echo "I did three million sines in " &
Int(Timer - before + 0.5) & " seconds!"
Loop
```

Note:

This is a script I use regularly to create CPU activity in a VM when I am teaching Vi-3 to students. It was actually written by VMware and is freely available on their website. You can find it on the download page from VMworld 2006; it comes from the Performance Troubleshooting Lab that ran there.

http://download3.vmware.com/vmworld/2006/labs2006/vmworld.06.lab04-PERFORMANCE-MANUAL.pdf

4. Right-click the cpubusy.vbs file within each VM, and choose Open in a command-prompt.

GOTCHA:

Don't double click the file, as this will produce dialog boxes from the wscript.echo command.

The idea of the script is that it shows very crudely how quickly the VM is doing the calculation.

Note: VMkernel CPU Load-Balancing/Scheduling

At this point you will find both VMs will generate alarms and alerts – and that both receive the same amount of CPU time. As you may recall from chapter 8 on Resource Monitoring, the VMkernel does a great job of scheduling these VMs so they don't run on the same physical socket or core. Figure 9.1 shows how both VMs are running on separate physical sockets (PCPU is 100.00 on both CPUs) – and although they are each consuming a lot of CPU time (VM3 is using 99.78 and VM1 is using 98.50), the %ready value is quite low (2.33 and 3.90 respectively). Effectively, the VMkernel is dynamically trying to stop contention occurring between the two VMs.

Figure 9.1

```
22:37pm up 6:19, 50 worlds; CPU load average
PCPU(%): 100.00, 100.00; used total: 100.00
CCPU(%): 0 us, 1 sy, 98 id, 1wa; cs/sec 77

Name  %USED…      %RDY
Vm3   99.78       2.33
Vm1   98.50       3.90
```

Creating Contention

One of the good features of VMware is the proportional share system. However, to see it "at work" we have to create some contention. As you might recall from this chapter, share system only operates if contention is occurring and the resource in contention is scarce. We can easily generate CPU contention by forcing two VMs to run on the same CPU socket, core or logical processor. This effectively stops the VMkernel scheduler from moving VMs from one CPU to another.

1. On your first VM, **Edit the settings**.
2. Click the **Resources Tab**.
3. Choose **Advanced CPU**.
4. Choose the **Run on processor(s)** option, and select just one CPU.

   Note:

   Repeat this process for the second VM running the CPU busy script, which will create contention. What you should see is both VMs start to process fewer "sines" per second. In other words, they are running slower. Figure 9.2 shows how only one physical CPU is now busy (the first CPU is almost idle at 2.05, whereas the second CPU is at 100.00). Additionally, we can see that each VM is getting roughly half of the CPU (49.95 and 49.83 respectively). This happens because by default every VM gets 1000 CPU shares each, and I have two running VMs which are very busy. That means there is a total of 2000 shares, with each VM getting 1000 each – resulting in 1000/2000, or ½ or 50%.

Lastly, notice how %RDY value has massively increased in size. Clearly CPU affinities can be very dangerous – as there is no warning from the Vi-Client about creating this kind of configuration. In fact, this kind of CPU contention is precisely what we are trying to avoid.

Figure 9.2

```
23:38pm, up 7.20, 50 Worlds, CPU load average:
PCPU(%): 2.06, 100.00; used total: 51.03
CCPU(%): 0 us, 1 sy, 98 id, 1 wa; cs/sec 96

Name  %USED…      %RDY
Vm3   49.95       52.40
Vm1   49.83       52.23
```

Creating Resource Pools

In this section I will create two resource pools, one called Production and the other called Test & Dev. We will give the production resource pool 3000 CPU shares and the test & dev resource pool 1000 shares. This won't be enough to stop the contention created by the CPU affinity feature – but will show that resource pools do offer an effective way of allocating more resources to one group of VMs than to another.

1. **Right-click the ESX host**.
2. Choose **New Resource Pool.**
3. Type a friendly name such as **Production.**
4. Under **CPU Resources and Shares**: select **Custom** and type **3000**.

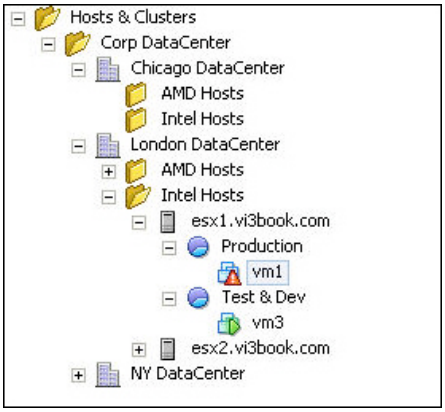
Next create a second resource pool:

5. Choose New Resource Pool.
6. Type a friendly name such as Test & Dev.
7. Under CPU Resources and Shares: select Custom and type 1000.

8. Next, drag-and-drop one VM to Production and the other VM to Test & Dev.

   Figure 9.3 shows the final configuration.

Figure 9.3



Note:

You should find that if you look at the cpubusy.vbs script that one VM is running much faster than the other, in my case VM1. Merely by calculating the CPU shares value I can take a good guess what the actual CPU utilization will be. We allocated a total of 4000 shares (Production's 3000 plus Test & Dev's 1000). This should mean that a production VM should receive 3000/4000 shares or ¾ or 75%, whereas the Test & Dev VM should receive 1000/4000 or ¼ or 25%. Figure 9.4 shows the current utilization. We can see that the physical CPU is still very busy. VM1 is receiving 74.92% of CPU time, whereas its partner in the other resource pool is receiving 25.05%. We can see that as VM1 receives more CPU time, its ready value has come down to 15.05%, whereas VM3 which gets less CPU time is running at 48.01%. In a production environment neither of these ready values would be regarded remotely as "acceptable." Remember, this is an academic exercise where we are deliberately creating contention to show the effectiveness of resource pools.

Figure 9.4

```
0:38am, up 9.20, 50 Worlds, CPU load average:
PCPU(%): 6.16, 95.05; used total: 51.06
CCPU(%): 4 us, 1 sy, 91 id, 3 wa; cs/sec 110

Name  %USED…      %RDY
Vm3   25.05       79.19
Vm1   74.92       27.92
```

Share Values on VMs within a Resource Pool

Lastly, you might wonder if you can create resource pools within resource pools. The answer is that you certainly can. You might also ask what happens to the share value set on VMs within a resource pool. The answer is they are still effective, too. So it is possible to give one VM more shares of a resource pool than another VM. For example, let's give the Production resource pool 3000 shares and Test & Dev resource pool 1000 shares. This would mean they would have 75% and 25% split of the CPU time. If within the Test & Dev resource pool one VM had 1000 CPU shares and another VM had 2000 shares – the 25% allocated to the Test & Dev resource pool would be divided accordingly. If contention occurred and CPU resources were scarce, this would mean our first VM would receive about 8% of the CPU time and our second VM would receive 16%. The moral of the story is that VMs get their resources not from the physical server directly, but indirectly from the resource pool. As a consequence, share values are calculated within the boundaries of the resource pool.

Admission Control: Insufficient Resources to Power On?

"Insufficient Resources" - It's not uncommon for people who are new to VMware ESX server to receive this message - indicating either insufficient memory or CPU required to power on a VM. It can happen with stand-alone ESX hosts and the VM's settings, ESX hosts with Resource Pools, and ESX Clusters with Resource Pools. It can be deeply frustrating because on first glance it often looks like you have plenty of resources. For example, you know you have an ESX host with 4 quad-core CPUs (16 cores altogether) and 32GB

or 64GB of RAM – and yet still your VM won't power on. The technical term that VMware uses to refer to this behavior is "Admission Control" – what are the resources required to power on your VM?

The source of these problems is always the same. It happens because a VM is given a reservation which is larger than the free resources currently available. Remember that reservations act as a guarantee to the VM or resource pool that a certain level of resources will always be available. In most cases it's impossible to allocate reservation which is greater than the resources that are free. This would be a bit like our example of an airline selling more seats than it has physically available (a policy which is actually common practice in most airlines!). So the "Insufficient Resources" message is like arriving too late at the airport and finding you failed to make your reservation. Planes and hotels are not easy resources to expand. So instead of adding more seats in the plane or building more rooms in the hotel your admission is refused.

This issue is best addressed by a couple of examples. We will create this error message – and then look at ways of resolving it.

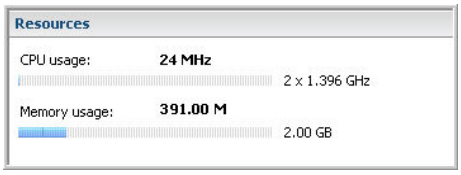
Example 1: ESX Host without Resource Pools

In this example I will take an ESX host with N amount of free memory – and then try to allocate more RAM to the VM as a reservation and see what happens. First, let's see what amount of resources I have available on ESX2.

1. Select an **ESX host**.
2. Click the **Summary Tab**.
3. View the amount of resources available in the resources pane.

   Figure 9.5 shows that this ESX host has 2GB of RAM, with 391.00MB currently in use. By my calculations that's about 1657MB of free unreserved memory.

Figure 9.5



4. Next take a VM, and power it off.
5. **Right-Click**, and choose **Edit Settings**.
6. Under the **Hardware Tab** set the limit on the **amount of RAM to,** say, 512MB.
7. Then click the **Resources Tab**, and set the **Reservation to,** say, 256MB.
8. **Next attempt to power on the VM.**

In the first instance this should be successful. But if you create another VM and go to power it with the same setting again and again, each one will reserve 512MB of RAM out of the available memory present.

Figure 9.6 shows the result of running more and more VMs which eat up the server amount of free resources. You might ask why VMware allows this to happen. Put simply, VMware has no idea how you will run your ESX host and what VMs you will have powered on at different times – also they have no idea that you might change the total amount of resources available, such as when you add an additional ESX host to a cluster.

Figure 9.6

Clearly here your only option would be to reduce the reservation value so there is enough memory to power on the VM.

Example 2: ESX Host with Resource Pools

One way of conceptualizing resource pools is to see them as taking a server and carving it up into slices of CPU or Memory. In this respect resource pools are logical divisions of resources, whereas servers represent a physical boundary. We might decide to take a physical server to create a resource pool with 25% of its resources and another resource pool that is 50% in size. This would leave 25% unallocated outside of the resource pool.

We can get a very similar situation as in Example 1 where we reserve to the resource pool 1GB memory, but try to allocate more than what is free in the pool. The critical thing which will determine if a VM will power on is if the resource pool has an "expandable reservation" setting. If the resource pool is expandable it can reach out and use the free 25% previously outlined. If the resource pool is not expandable then it is "locked" within the confines of its own reservation. So perhaps the production resource pool would get 50% of the RAM in its reservation, with the resource pool set to be expandable, but the Test & Dev resource pool would only get 25% of RAM, set to non-expandable. Once all the memory allocated to Test & Dev had been used from the reservation a VM would not power on.

In the following example I will create a similar power-on error using a resource pool. I will reserve to the Test & Dev resource pool 1GB of RAM and then modify the settings of my VM located in the resource pool more RAM than is available. I will leave the default setting in place which allows the reservation to be expandable, then I will power on the VM to see what happens. After that I will power off the VM, and I will then change the resource pool to non-expandable.

1. Right-Click the Test & Dev Resource Pool and choose Edit Settings.
2. Under Memory Resource, move the Reservation slider to 1024MB and Click OK.

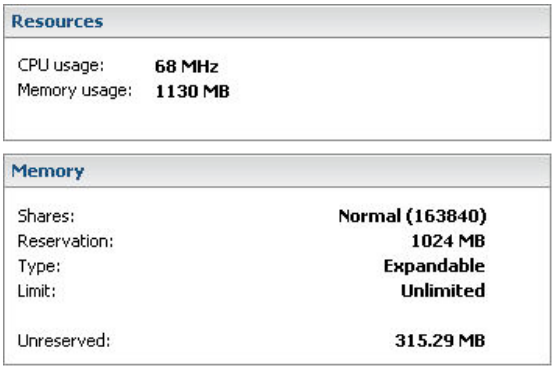
Next we will modify a VM that uses practically all the resources of the pool.

3. Next, take a VM and power it off.
4. Right-Click, and choose Edit Settings.
5. Under the Hardware Tab set the limit to 512MB.
6. Then, click the Resources Tab, and set the Reservation to 256MB.
7. Power on the VM.
8. Define other VMs in this way and begin to power them.

   Note:

   Figure 9.7 shows the "Summary Tab" on a resource pool – notice that memory usage is 1130MB, but the reservation is only 1024MB. The reason the VMs were able to power on was that free unreserved RAM of 315.29. The VM was allowed to use this free and unreserved RAM because the resource pool was "expandable."

Figure 9.7



9. Now power off the VM.
10. Right-Click the Resource Pool, and Choose Edit Settings.
11. Under Memory Resource, remove the tick next to Expandable Reservation.

You should find now that you get the same message we saw in Figure 9.6. As you went to power on VM1, VM2, VM3, and VM4 they began eating up the pools reservation in blocks of 256, 512, 768, and 1024. When you went to

power on VM5 it would fail as there would be insufficient RAM in the resource pool left to meet your reservation. It would be like a 4 seat plane – and you were the 5th person arriving. There would be insufficient resources (seats) to make your reservation.

We can apply this to our airline analogy. The resource pool is like the Hollywood star's reservation to fly him and ten members of the staff. When he arrives at the airport he decides his personal stylist and hairdresser also need to come. At the check-in desk, the clerk recognizes that he's a premium flyer, and his reservation is expanded. Before she smiles sweetly and says that's fine Mr. X, she first must check that there are two extra seats free in business class. In my case, I am a less privileged person. When I arrive at the check-in desk the clerk checks my status and decides I'm an ordinary Joe. My reservation is not marked expandable – and my mother-in-law has to wait in the booking hall. She doesn't even look to see if there are spare seats on the flight. She knows I'm not business class, and the request to expand my reservation is politely declined. In the UK, we would call this "Computer says no….."

If you are facing this scenario there could be a number of solutions to the problem. The list below is ordered by the change that would require least impact on the rest of the system. The first would only require a change to a VM that is affected by the lack of RAM, the last would require all VMs to be powered off, to give them less generous reservations of RAM:

- Decrease the amount of memory reserved for the VM
- Increase the reservation of RAM to the resource pool
- Enable the Expandable Reservation option
- Move the VM to a resource pool that has greater resources
- Decrease the amount of memory reserved for other VM's in the same resource pool

Summary

In this chapter I have explained the different controls you have available such as limits, reservations, shares, CPU affinity, and resource pools. I have also outlined some of the advantages and disadvantages of each method – and given

you a practical example of using resource pools and creating contention to demonstrate the effectiveness of them and the proportional share system. Lastly, I showed how VM reservations of memory and CPU can cause a VM not to power on – referred to by VMware as "Admission Control." Additionally, I gave some guidelines on how to resolve these power-on issues.