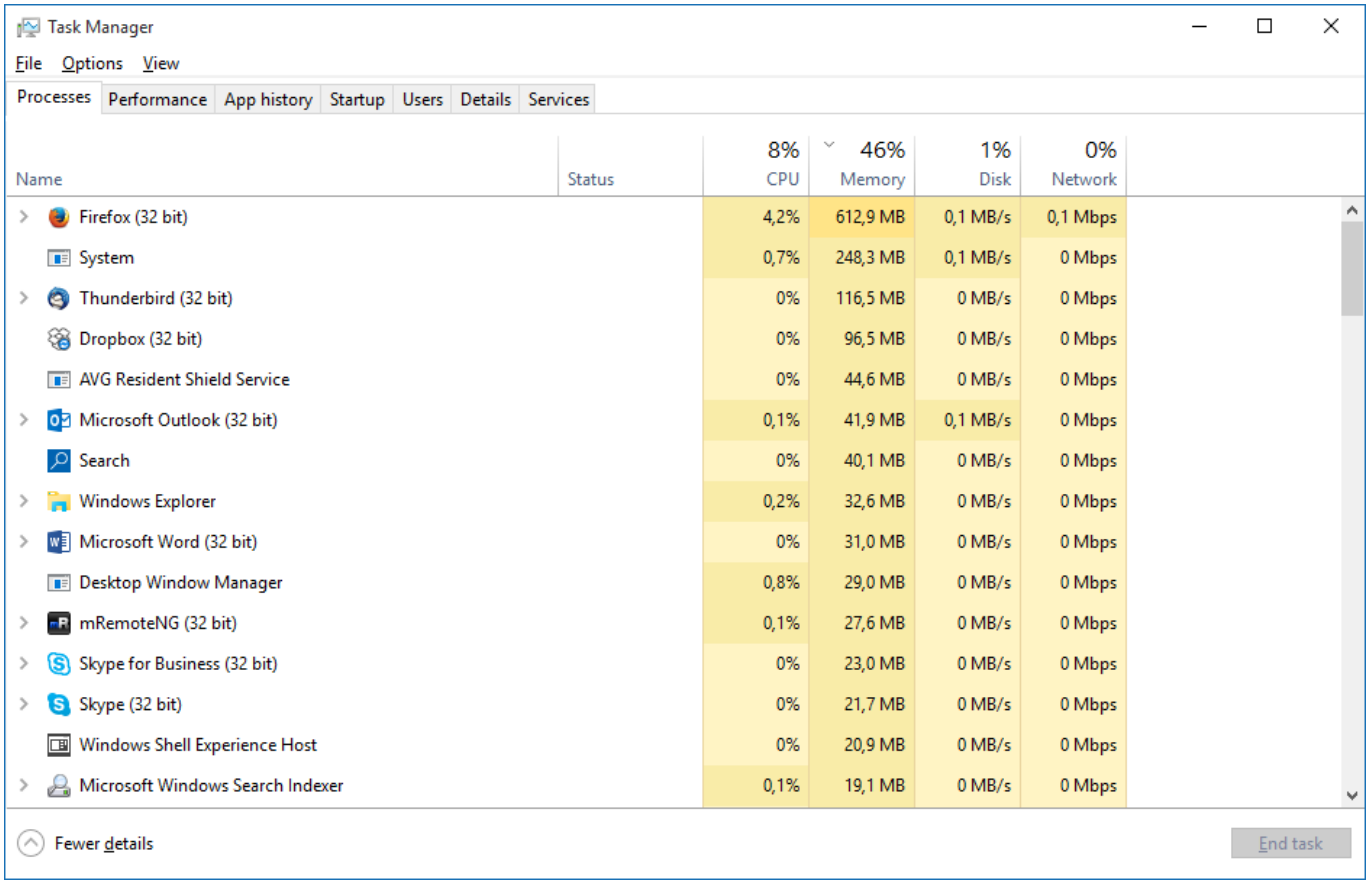
Hyper-V Virtual CPUs Explained

Not clear on what happens when you assign CPUs to a virtual machine? The good news is that you are not alone! Even better news is that you about to get an explanation!

First thing you must know is that physical processors are never assigned to specific virtual machines. Assigning two virtual CPUs to a virtual machine does not mean that Hyper-V reserves two cores out of the physical pool and permanently assigns them to that virtual machine. That's not what any hypervisor does. At least Hyper-V doesn't.

Understanding operating system processor scheduling

Take a look at a screen shot of my Task Manager screen.



Screenshot of my Task Manager

When computers never came with multiple CPUs or CPUs with multiple cores, we knew that computers couldn't really multitask. There was one CPU with one core, so there was only one possible thread of execution.

Task Manager then looked pretty much like Task Manager now. You had a long list of running processes, all of them with a metric indicating what percentage of the CPUs time they were using. Each line item you see is a process (in the recent Task Manager versions, a process group).

A process might consist of one or many threads. A thread is nothing more than a sequence of CPU instructions.

What happens is that the operating system would stop a running thread, preserve its state, and then start another thread. After a bit of time, it would stop that thread, preserve its state, and then start another thread, and so on. All this is pre-emptive, which means that it's the operating system that decides when a new thread will run. The thread can beg for more CPU time, and priorities can be set that affect where a process goes in line, but the OS is in charge of thread scheduling.

The only difference today is that we have multiple CPUs with multiple cores in practically every system (and don't forget about hyper-threading in Intel processors), so operating systems can actually multi-task now.

Applying these concepts to the hypervisor

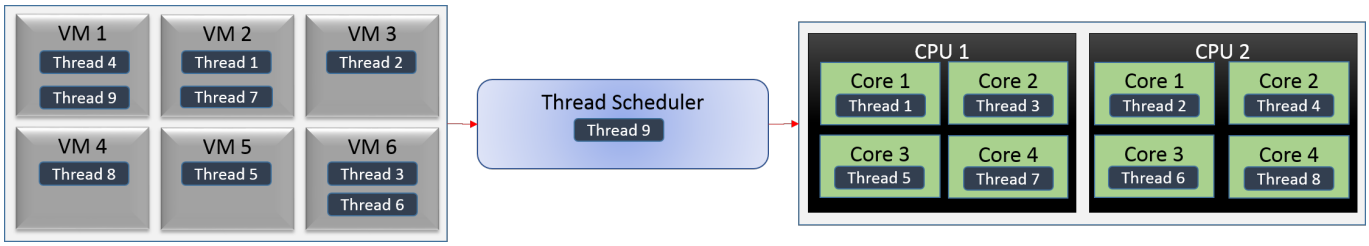
Because of its role as a thread manager, Windows can be called a "supervisor" (very old terminology):

"A **supervisory program** or **supervisor** is a computer program, usually part of an operating system, that controls the execution of other routines and regulates work scheduling, input/output operations, error actions, and similar functions and regulates the flow of work in a data processing system."

Hyper-V is a hypervisor:

"A system that manages supervisors that manage processes that are made up of threads."

Task Manager doesn't work the same way for Hyper-V, but the same thing is going on. There is a list of partitions, and inside those partitions are processes and threads. The thread scheduler works pretty much the same way.



Hypervisor Thread Scheduling

The image presented above is necessarily an oversimplification, as it's not simple first-in-first-out (NUMA plays a role). There are always going to be a lot more than just nine threads going at any given time. They'll be queued up in the thread scheduler.

Processor affinity

We know that we can affinitize threads in Windows so that they always run on a particular core or set of cores. As far as I know there's no way to do that in Hyper-V with vCPUs. Dedicating a thread to a core is not the same thing as dedicating a core to a thread. You can't prevent a core from running other threads in the Windows world, which is what many people really want to try to do.

How does thread scheduling work? The simplest answer:

Hyper-V makes the decision at the hypervisor level, but it doesn't really let the guests have any input. Guest operating systems decide which of their threads they wish to operate.

Really understanding this topic requires a fairly deep dive into some complex ideas, and that level of depth is not really necessary for most administrators.

Affinity aside, you never know where any given thread is actually going to execute. A thread that was paused to yield CPU time to another thread may, when it's resumed, be assigned to another core.

When you see an application consuming exactly 50% of a dual core system and each core looks like it's running at 50% usage, that indicates a single-threaded application. Each time it is scheduled, it consumes 100% of the core that it's on. The next time it's scheduled, it goes to the other core and consumes 100% there. When the performance is aggregated for Task Manager, that's an even 50% utilization for the app. Since the cores are handing the thread off at each scheduling event and are mostly idle while the other core is running that app, they amount to 50% utilization for the measured time period. If you could reduce the period of measurement to capture individual time slices, you'd actually see the cores spiking to 100% and dropping to 0% (or whatever the other threads are using) in an alternating pattern.

What we're really concerned with is the number of vCPUs assigned to a system and priority.

What number of vCPUs we select actually means

You can't assign more vCPUs to a virtual machine than you have physical cores in your host. So,a virtual machine's CPU count means the maximum number of threads that it is allowed to operate on physical cores at any given time. Here, I can't set that virtual machine to have more than four vCPUs because the host only has four CPUs. Therefore, there is place for a fifth thread to be scheduled. If I had more cores on my system and left this VM at 4 vCPUs, then it would only ever send a maximum of four threads up to the hypervisor for scheduling. Other threads are kept in the guest's thread scheduler (the supervisor), waiting their turn.

So why can we assign more total vCPUs to all VMs than physical cores?

This is no different than the fact that we've got 50+ processes running on a quad core laptop. We can't actually run more than four threads at a time, but we'll always going to have far more than four threads scheduled. Windows is so good at this (most of the time) that most people don't even pause to consider just what's going on. Your VMs (supervisors) will bubble up threads to run and Hyper-V (hypervisor) will schedule them (mostly) the same way that Windows has been scheduling them ever since it outgrew cooperative scheduling in Windows 3.x.

The proper ratio of vCPU to pCPU/Cores

In the generic sense, this has no answer. Some people say 1:1 – and can do it but it's wasteful. I could run my current desktop configuration on 64-core server but I wouldn't see much performance difference because almost all my threads sit idle almost all the time. If something needs 0% CPU time, giving it its own core does nothing. Later, the answer was upgraded to 12 vCPUs per 1 physical core. And then the recommendations went away.

It was probably a good rule, but think about it. You know that mostly, operating threads will be evenly distributed across whatever hardware is available. So the amount of physical CPUs needed doesn't depend on how many virtual CPUs there are. It's entirely dependent on what the operating threads need. And, even if you've got a bunch of heavy threads going, that doesn't mean their systems will die as they get pre-empted by other heavy threads. It really is going to depend on how many other heavy threads they wait for.

The CPU is a binary device; it's either processing or it isn't. Here's the kicker: Every single time a thread runs, no matter what it is, it drives the CPU at 100% (power-throttling changes the clock speed, not workload saturation). The 100% or 50% or whatever number you see is completely dependent on time measurement. 100% means that the CPU was completely active across the measured span of time. 50% means it was running a process half of that time and was idle the other half of that time.

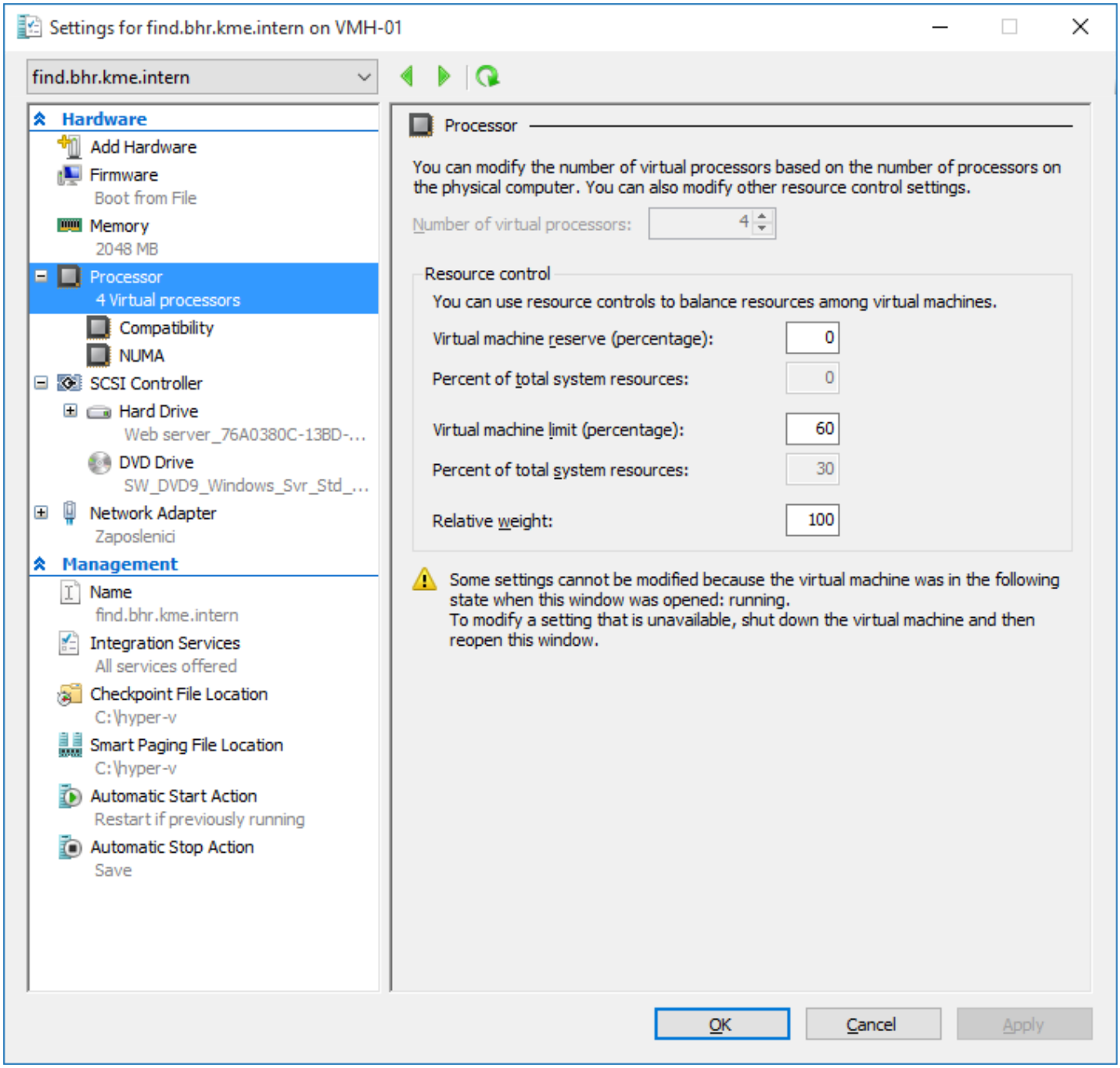
This means is that a single thread can't actually consume 100% of the CPU the way people think it can, because Windows/Hyper-V will pre-empt it when it's another thread's turn. You actually can have multiple "100%" CPU threads running on the same system.

The problem is that a normally responsive system expects some idle time, meaning that some threads will simply let their time slice go by, freeing it up so other threads get CPU access more quickly. When there are multiple threads always queuing for active CPU time, the overall system becomes less responsive because the other threads have to wait longer for their turn. Using additional cores will address this issue as it spreads out the workload.

This means that if you really want to know how many physical cores you need, you need to know what your actual workload is going to be.

Reserve and Weighting (Priority)

Don't mess with CPU settings unless you really understand what's going on. Let the thread scheduler do its job. Just like setting CPU priorities on threads in Windows can get initiates into trouble in a hurry, messing with hypervisor vCPU settings can throw a wrench into the operations.



You can’t assign more vCPUs to a virtual machine than you have physical cores in your host.

Virtual machine reserve (percentage)

The first textbox represents the percentage that we want to set, and its actual meaning depends on how many vCPUs we’ve given to the VM. In this case, I have a 4 vCPU system on a quad core host, so the two boxes will be the same. If I set 10 percent reserve, that’s 10 percent of the total physical resources. If I drop this down to 2 vCPU, then 10 percent reserve becomes 5 percent physical. The second textbox (which is disabled) will be calculated for you as you adjust the first box.

The reserve is sort of a minimum. If the total of all reserve settings of all virtual machines on a given host exceeds 100%, then at least one virtual machine isn’t going to start. If a VM with a 20% reserve is idle, then other processes are allowed to use up to 100% of the available processor power until such time as the VM with the reserve starts up. Then, once the CPU capacity is available, the VM with the reservation will be able to dominate up to 20% of the total computing power. Because time slices are so short, it’s effectively like it always has 20% available, but it does have to wait like everyone else.

Virtual machine limit (percentage)

Now that we understand the reserve, we can understand the second textbox – limit. It's a resource cap. It keeps a greedy VMs under control.

Relative weight

The third textbox is the weight. This is relative. Every VM set to 100 (the default) has the same pull with the scheduler, but they're all beneath all the VMs that have 200, and above the VMs that have 50. If you're going to tinker, this is safer than messing with reserves because you can't ever prevent a VM from starting by changing relative weights. Weight means is that when a bunch of VMs present threads to the hypervisor thread scheduler at once, the higher weighted VMs go first. That's all.

What about Hyper-Threading

Hyper-Threading is an Intel-specific technology that lets a single core process two separate instructions in parallel (called pipelines). There is one problem: the pipelines run in lockstep. If the instruction in pipeline one finishes before the thread in pipeline two, pipeline one sits and does nothing. That second pipeline shows up as another core. But Hyper-Threading should not be counted toward physical cores when considering hypervisor processing capabilities.