Tips for optimizing performance in virtual environments

# FASTER VIRTUE

Tetastock, Fotolia

Virtual performance tuning is a lot like ordinary performance tuning – but not exactly.

**BY FEDERICO LUCIFREDI**

Since the rise of low-cost virtualization in 1999 with the first release of VMware Workstation, the public has rallied around the many benefits of virtualization. However, users might often wonder how to minimize the performance penalties they are paying when they work with virtualization technology.

The exact shape of virtualization's performance footprint has evolved as the field has matured. When 400MHz processors were first appearing on the market, the limiting factors posing an obstacle to widespread use of virtualization were CPU speed and RAM. This situation improved as Moore's law continued its inexorable march, providing both the processing power and the memory space sufficient for multiple virtual machines to run at once on the same hardware, and thereby opening the way for the flourishing server virtualization market.

A second performance challenge arises from the intrinsic ability of virtualization to allow overcommitting of physical resources. Assigning more (virtual)

## Before Deployment

Even before a single bit is rolled out to your virtual and physical hardware, several performance considerations enter the picture through deployment planning. The first question is, "What workload should be virtualized?" Although it is technically possible to virtualize almost any service, planners needs to choose with an eye to performance: Quite obviously, a service that is maxing out a particular system resource (network I/O, disk I/O, CPU) makes for a poor virtualization candidate. One of many ways to think about virtualization is as a trade-off between spare capacity and operational flexibility. If spare capacity is absent, virtualization is not going to help you out of your troubles.

Even as current virtualization solutions sometimes deliver near-hardware performance, in scenarios that aggregate multiple virtual workloads on the same physical host, you must take care that none of the fundamental performance metrics of the physical asset are exceeded by the *combined* use of the hosted VMs. If you choose to allow overcommitting of physical resources, you should consider the total throughput requirements of the workloads committed to a given piece of hardware at peak load, as these workloads are *sharing* that 90 percent of physical performance that your vendor is promising.

VM migration and an intelligent orchestration facility to manage it can address peak-load collisions effectively and can simplify one part of the planning process at the expense of another – namely implementation of the resource management system itself. Even when migration is part of the deployment process, the constant performance objective coloring the operational picture is to ensure that the combined requirements placed on a single host do not exceed the capacity on either the disk, network, or CPU axis. Your excitement and enthusiasm for virtualization should not cloud the obvious facts: Workload consolidation allows better use of existing hardware capacity, but no new resources are magically "created" by the virtualized setup.

processors to a set of virtual machines than the physical machine happens to have is an acceptable choice under a low service load, but as one or more of the hosted workloads experiences peak usage, a dynamic resource load-balancing scheme is required. Virtual machine migration, termination of VMs hosting lower priority tasks, or equivalent approaches must be orchestrated through a supervising logic to ensure that the service level is guaranteed, even as the performance "insurance" of physical machine separation is removed.

A third performance challenge rises from the need to juggle workloads to tackle performance measurement in a virtualized environment: Adding a virtualization layer to the complexity of today's system integration layouts increases the number of factors that the site administrator needs to manage for a successful and efficient deployment.

In this article, I outline some vendor-neutral tips for improving performance in virtual environments.

## Benchmarks

The original Xen team [1], VMware [2], and the multiple Xen vendors have produced some excellent material describing the performance characteristics of the hypervisor du jour. Without delving into too many details, as a rule of thumb, you can expect that a workload suitable for virtualization, running as the sole VM on a well-tuned hypervisor/hardware combination, will deliver 85% or better of the same hardware's native performance.

The mindset you should adopt when looking at a new virtualization deployment is that you are looking to trade CPU capacity for one or more of virtualization's advantages (server consolidation, hardware independence, workload migration, snapshot/replay of state, etc.). From that viewpoint, you will drill down to the specific needs of the workload, but always keep in mind that you are trading CPU for convenience.

## Avenues for Better Performance

One of the prominent decisions you will make in your quest for "90 percent performance" is whether to include in your solution a kernel that has been paravirtualized with technologies such as VMware's Virtual Machine Interface (VMI) or Microsoft's hypercall adapter [5]. These technologies provide for a hypervisor-specific way to accelerate certain aspects of the guest kernel's operation. The system call entry and return paths, in particular, are significantly accelerated, and virtualization's memory management overhead is reduced in a way that is significant for some workloads [6]. Paravirtualized device drivers enable conceptually similar hypervisor integration for operating system kernels that have not been otherwise optimized to work in a virtualized environment.

A key consideration when tackling virtualization performance is that the old physical performance lessons still apply … if you know how and where to look. The performance tuning process itself is unchanged: When faced with a problem, you use tools to take actual tangible measurements of the situation, which you then compare with your operational baseline. Afterwards, locate the bottlenecks this data highlights and eliminate them, together with any contention among your virtualized guests. The difference is that, in the traditional optimization process, you are looking at a single host. Now you have to consider both the workload guest, the virtualization host/hypervisor, and the interaction with other guests that might be running on the same physical iron. To do so, you need a new set of tools that enable you to form an overall impression by studying the performance, looking *across* guests, within a host, and within a guest. Virtualization adds another layer to the alchemy of the performance tuning art, but it does not invalidate the old craft, as long as the practitioner is aware of the new "knobs" that virtualization introduces in the additional abstraction layer.

## Tooling Considerations

The tool chest is expanded in a way that depends on the virtualization technology of your choosing; however, the patterns are the usual ones: Our old friend *top* is supplemented by virtualization-aware variants such as virt-top (Figure 1) or esxtop. One factor simplifying the picture of open source virtualization is that, because most of the F/OSS tools are implemented against libvirt, they are intrinsically able to operate with Xen, KVM, and potentially some container solutions without any implementation changes. As a result, virt-top (which provides disk throughput and network traffic data along with CPU measurements) and similar tools, like virt-df, work on a variety of virtualization platforms.

One needs to be careful with program counters when using tools that are not virtualization-aware: Because these tool measure the cycles and performance of the physical CPU as a whole (rather than the "virtual CPU" slice assigned to a given VM), the numerical results can be off altogether. In most cases, the trending between different situations is correct, but the specific numbers will not reflect actual values.

Another problem occurs with time: Aside from the well-known issues of clock-skew in virtualization, there is no simple way for *time* to tell if the CPU share assigned to a VM has significantly changed at the hypervisor level. As a new VM starts, a previously running VM on the same system internally shows that 90 percent of the CPU usermode allocation is currently spent in the workload; however, an actual measure of the

### Container Virtualization

As you choose your virtualization platform, do not discount the potential of operating system containers. Although container virtualization systems such as the open source OpenVZ are considerably less hyped than full virtualization solutions, container architectures are available for just about every *nix platform. In the most general terms, containers offer a lesser degree of isolation than hypervisors provide, in that they leverage the operating system's process abstraction and they might be limited to running a single kernel version (or one that has been modified for such use). Nonetheless, modern container offerings make a perfectly palatable solution where the operational needs match the design.

Vendor studies show that containers are marginally faster than full virtualization [3], but I recommend taking some time to examine whether it is actually possible to achieve a dramatically better result for your specific workload and operational requirements. If such a trade-off is sufficiently significant, go for it; otherwise, you should default to the full virtualization solution, as it is generally more flexible.

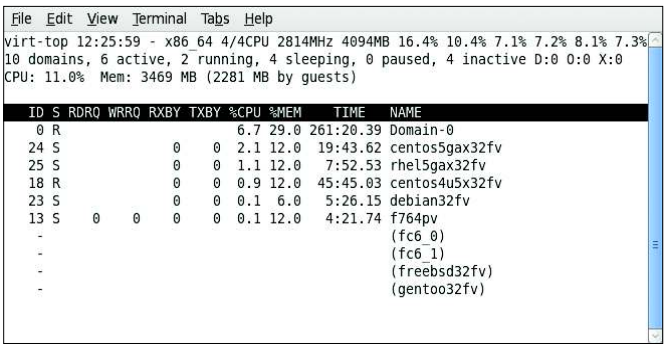


**Figure 1: The virtualization-aware virt-top is modeled on the classic Unix top utility.**

workload performance shows that it is now progressing at half the original rate and taking twice as long to complete. Yet, the guest-based measurements say that the half-as-fast workload is still taking the same share of its virtual CPU as it had before: From the point of view of the guest, it is literally as if the CPU was swapped in flight with a less powerful one. Because this is not something expected by most programmers, such tools might fail to produce correct results when faced with this situation in a guest environment.

Although the details of the tools themselves are strongly dependent on the virtualization architecture, the coding strategies are few and very clearly defined: Just as practitioners of performance optimization need to be familiar with universal operating system concepts (buffering, caching, swapping, out of memory behavior, process states, etc.), regardless of whether the operating system is Linux, Solaris, Windows, or otherwise, those tuning virtualized environments need to be familiar with the few alternative architectures that are used to achieve isolation. Knowing how code execution and memory access is virtualized and how devices are mapped in your specific implementation is necessary if you want to understand and diagnose unusual behavior (e.g., increased interrupt count, altered timing, modified RAM footprint, etc.) [7]. A half a day spent familiarizing yourself with such details will pay back handsomely in time saved later when faced with complex, confusing, and misleading real-world scenarios. The ability to debug interactions between the guest and the virtualization layer is the most important tool you need to acquire: Most feedback loops and other degenerate scenarios are only apparent if you know how the magic works.

## Best Practices

As I mentioned earlier, virtualization is the art of trading off one facility (the CPU) for an otherwise unavailable set of functionalities. If your workload saturates the CPU, you should think twice before planning to virtualize it. In addition to this all-important criterion are some other suggestions that will help you get the most from your processor.

The first task is to examine whether it is possible to "pin" a dedicated CPU (or a core) to a specific virtual machine, effectively creating a mapping between that VM's virtual CPU and a dedicated physical processor. Doing so drastically reduces cache trashing, and as any performance maven knows, modern processor performance is tied to cache hits more than to any other single factor. If this is not possible, it is generally wiser to at least assign the same number of CPUs to all VMs hosted on a given machine – even when overcommitting. This strategy derives from the inherently simpler picture that the hypervisor's thread scheduler will have to contend with if the CPUs are balanced. Similarly, avoid assigning more virtual CPUs than are strictly necessary: If your workload cannot make effective use of multiple cores, avoid virtual SMP (Symmetric Multiprocessing) configurations – the additional virtual CPU still requires interrupts and creates overhead just by being present.

Of course, if your virtual guest is indeed SMP-enabled, you will want to consider tuning affinity *within* the guest to prevent too many processor migrations from adversely affecting performance. Make sure you are always using the right kernel flavor: SMP for multiple cores and uniprocessor for a single virtual CPU. The uniprocessor kernel will not make use of additional virtual CPUs, and the SMP kernel carries additional overhead, which is wasteful when a single processor is in use. Another suggestion is to remember that CPU affinity can be assigned for IRQ requests as well as threads under the Linux kernel: Consider offloading the interrupt servicing to a dedicated processor or spreading it uniformly where interrupt-intensive devices (such as multiple network cards) are present in your system.

Some virtualization architectures cleverly detect kernel idle loops and reduce the VM's scheduling priority, This strategy can affect performance, and you will

### Hardware Testing with VMmark

Numbers provided by your trusted vendor are well and good, but even the most reputable of third-party benchmarks won't be a perfect match for your hardware choices. Ultimately, you will need to assess your actual target environment. Currently, VMware's VMmark [4] is a popular choice for virtual performance benchmarking. First released in 2006 and now at version 1.1, VMmark differs from one-workload benchmarks by creating a single measurement for the virtualization environment out of a variety of workloads consolidated on a hardware host and running concurrently in separate virtual machines. VMmark refers to the measured unit of work performed by a collection of virtual machines as a "tile."

If you feel like studying your virtual systems with VMmark, start by downloading the appropriate bits from the VMware site, including the VMmark toolkit and one or more workloads, some of which are neatly pre-packaged as virtual appliances. Getting VMmark running on your machines is not as straightforward as rolling out other VMware products, so you will want to head straight for the */docs* directory in your VMmark package and start reading through the Benchmarking Guide. The Guide contains detailed checklists that will help you navigate through the maze of required and optional steps needed to set up the benchmark.

Once the hypervisor you want to test is running on your benching hardware, you will need to select the test workloads. Although some test loads are effectively supplied ready "out of the box" in their virtual appliance, others require a more convoluted set-up (because of licensing limits on non-free components). Running a full virtualization benchmark correctly is not trivial, and will make considerable hardware allocation as additional clients are needed to drive each "tile."

Anzeige
wird
separat
angeliefert

want to know the exact mechanics under which this occurs in your system to determine whether it is beneficial or harmful to the workload.

The availability of shared memory pages between multiple identical guests is a very significant factor to consider when choosing how your workload is hosted: If multiple VMs are running on the same host, you can gain a non-trivial advantage by choosing to deploy the same OS image for all the VMs, irrespective of any workload differences. If you use the same image for all VMs on an architecture on which shared memory pages are well implemented, you will achieve a significant reduction in the allocation of actual physical RAM because the multiple copies of those identical OS pages are loaded in memory only once.

It is a good idea to spend some time tuning virtual memory allocation for the needs of the workload: You will want to provide your virtual systems with a comfortable amount of RAM, which will minimize, and possibly eliminate, the need for swapping. Page faults in virtual environments affect performance more than in physical systems, and you should avoid them as much as possible. It is, however, also advisable to avoid assigning excessive amounts of memory, in that this complicates the hypervisor's memory management work, which can result in complex swapping situations if multiple overcommitted VMs are running simultaneously and the hypervisor must force one to yield resources.

Large page support can also improve the performance of workloads that would benefit from a similar setup in non-virtual environments; benchmark your load and determine whether the change is helpful or detrimental in your case. Finally, a significant number for Linux guests is 896MB: Memory pages up to this RAM size are mapped directly into the kernel space, whereas those beyond this boundary require a slightly more involved addressing scheme, an unnecessary overhead if you can possibly avoid it.

Mass storage benefits from simplification just as other components do, and you should avoid complex layouts when they are unnecessary. One example seen in the field is significantly degraded performance with the use of LVM volumes simultaneously on the guest and on the host. LVM is hardly necessary for the guest because the guest's virtual disks are inherently resizable and can be structured on different physical storage media. Swapping should be avoided as a matter of course, but when you can't eliminate it, it makes sense to optimize it by directing I/O activity to different physical disks.

Solid state units are great candidates for fast swap, but one should also remember that, because of the properties of zone bit recording (ZCAV), the outer tracks of a standard hard drive provide much higher raw data transfer rates than the inner tracks. As you lay out your physical partitions, keep this fact in mind and spread the layout to multiple disks if you can. Conversely, you will want to avoid specific I/O scheduler choices within your guests: Their built-in assumptions will most likely not hold in a virtual environment. As a result, it is often best to default to the NOOP scheduler for the guests' kernel because the duty of optimizing read/write performance falls to the host and the complexity of more sophisticated schemes at the guest level will not be helpful and might indeed be harmful.

To ensure optimum performance, defragment disks, both virtual and physical. Just proceed from the guests outward to the hosts, and take into consideration the properties of snapshots in your particular system. Incidentally, as of this writing, several vendors recommend SCSI virtual disks as offering the best performing I/O subsystem: The EIDE bus, even a virtual one, is limited to a single transaction at a time.

A study of network performance would require another full article. Some common pitfalls include the use of a virtual driver that is sub-optimal (the typical example is the use of VMware's vlance instead of the more optimized vmxnet) or the unrecognized failure of duplex auto-negotiation. Performance tuning of the network side of virtualization is evolving rapidly with the appearance of hardware-assist technologies such as Virtual Machine Device Queues (VMDQs), which offload the burden of network I/O management from the hypervisor into NIC hardware that supports multiple parallel queues.

Because much attention is paid to the low-level details, higher level decisions, such as what network protocols to use for data storage, warrant significant consideration, too. Recent results show that iSCSI in both software and hardware implementations and NFS are largely comparable solutions [9] , with the more expensive Fibre Channel still standing out as providing significant improvement.

## Conclusions

Carefully choose a workload, simplify the configuration of the virtual machine it will run within, and proceed to performance characterization and tuning. These simple steps are but a start; many specific details inherent to your chosen virtualization technology will have to enter the picture as you test and measure to achieve your target performance.

After you repeat the process a few times, you will learn to value predictable VMs that can be accommodated with static resource allocations, in that they are much easier to plan for than those whose resource usage expands and contracts unpredictably; such guests make poor neighbors to other workloads. ■

### INFO


[1] Xen and the Art of Virtualization: *http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf*

[2] A Performance Comparison of Hypervisors: *http://www.vmware.com/pdf/hypervisor_performance.pdf*

[3] Container-Based Operating System Virtualization: *http://www.cs.princeton.edu/~mef/research/vserver/paper.pdf*

[4] VMmark: A Scalable Benchmark for Virtualized Systems: *http://www.vmware.com/pdf/vmmark_intro.pdf*

[5] Hypervisor Functional Specification: *http://www.microsoft.com/downloads/details.aspx?FamilyId=91E2E518-C62C-4FF2-8E50-3A37EA4100F5&displaylang=en*

[6] Performance of WMware VMI: *http://www.vmware.com/pdf/VMware_VMI_performance.pdf*

[7] A Comparison of Software and Hardware Techniques for x86 Virtualization: *http://www.vmware.com/pdf/asplos235_adams.pdf*

[8] VProbes Programming Reference: *http://www.vmware.com/pdf/ws65_vprobes_reference.pdf*

[9] Comparison of Storage Protocol Performance: *http://www.vmware.com/files/pdf/storage_protocol_perf.pdf*