THE VIRTUALIZATION SPECTRUM FROM HYPERTHREADS TO GRIDS[†]

Neil J. Gunther
Performance Dynamics Company, Castro Valley, California, USA
www.perfdynamics.com

Modern virtual machines (VMs) can be opaque to conventional performance management tools because VM technology has surpassed standard measurement paradigms. We attempt to ameliorate that problem by (1) observing that disparate types of VMs lie on a discrete spectrum bounded by hyperthreading at one extreme and GRID-like services at the other, and (2) recognizing that poll-based scheduling is the common architectural element in many VM implementations. The associated polling frequency (from GHz to μHz) positions each VM in its respective region of the VM-spectrum. Several case studies are analyzed to illustrate how this insight can make VMs more visible to performance management techniques.

1 INTRODUCTION

Virtualization remains a hot topic because of the opacity of virtual systems from the standpoint of conventional performance management. These difficulties are further exacerbated by the appearance of so many disparate implementations employing the adjective *virtual* [See e.g., Sin04]. Moreover, modern computer architectures that create virtual resources and services out of physical resources are quite distinct from the more familiar virtualization paradigms e.g., virtual memory or virtual storage.

This paper proposes a more unified picture of modern virtualization by recognizing that many of these apparently disparate forms of virtualized resources or virtual machines (VMs) can be considered to lie on a discrete spectrum—the *virtual machine spectrum* or *VM-spectrum*—comprised of three principal regions:

1. Micro-VMs: represented by the hyperthreaded multicore processors discussed in Section 3.
2. Meso-VMs: represented by virtual machine monitors and hypervisors discussed in Section 4.
3. Macro-VMs: represented by the GRID services and peer-to-peer (P2P) architectures in Section 5.

[†] Copyright © 2006 Performance Dynamics Company and CMG, Inc. Accepted for publication in the Proceedings of the Computer Measurement Group (CMG) 32nd International Conference, Reno NV, December 3–8, 2006. Version of September 11, 2006

The inclusion of GRIDs and P2P under the umbrella of virtualization is an unusual step, but our intent is to use the VM-spectrum both as a classification scheme to organize the discussion of an otherwise bewildering array of VM architectures, and as a quantitative framework for explaining previously reported performance anomalies in a variety of controlled measurements on VM systems.

Most previous discussions have tended to organize VM capacity planning issues according to whether they are implemented in *hardware* or *software* [See e.g., Joh03, DBK03, Fri03, Bra05, Fer05]. Macro-VMs have not featured in such discussions, as far as I am aware. Consequently, the inability to achieve full virtual capacity in hyperthreaded hardware and anomalous performance outcomes in software hypervisors, have been presented as distinct performance effects.

The VM-spectrum paradigm, however, views these effects as arising from an architectural feature that is common to both hardware and software implementations; a form of scheduling which we define in Section 2.2 as *proportional polling*. In particular, it provides a simple explanation for the observed *Missing MIPS* problem in Sect. 3.2.2. The VM-spectrum also leads to the notion that performance management of modern VMs is a function of the time and distance scales on which their respective polling mechanisms operate.

The paper is organized as follows. Section 2 defines the rationale for the VM-spectrum and its three principal regions. Each of these VM-regions is examined in detail, starting in Section 3 with the micro-VM scale. The best known VM implementations in this region are virtual processors or hyperthreaded CPUs. Virtual processors also constitute the first example of a polling-based scheduler operating in the GHz to kHz frequency range. The associated performance case studies and recommendations are presented in Section 3.2. Section 4 moves up the VM-spectrum (Fig. 1) to slower polling meso-VMs represented by virtual machine monitors or hypervisors. These VMs constitute the second example of a polling-based scheduler operating in the kHz (kilohertz) to mHz (millihertz) frequency range. The associated performance analysis case studies and recommendations are presented in Section 4.2. Section 5 discusses the slowest polling macro-VM services represented by GRIDs and P2P networks. At this scale, different types of state information are collected via polling mechanisms which can have operating periods in the range of days! Obviously, these frequency ranges can have a significant impact on the performance and scalability of virtual services. The associated performance analysis case studies are presented in Section 5.2. Section 6 presents conclusions and recommendations.

2 VIRTUALIZATION SPECTRUM

Virtualization is about creating illusions. In particular, modern computer systems are now sufficiently powerful to present users with the illusion that one physical machine is really multiple virtual machines, each one running a separate instances of a different operating system (OS). This is one reason for the resurgence of interest in virtualization technologies. In the interests of space, we forego any review of the history or relative merits of various VM projects and products, which are adequately discussed elsewhere [See e.g., Sin04, Fer05, and references therein].

As many authors have already noted, the idea of creating virtual resources e.g., software emulators and virtual memory, is not new. For the purposes of this paper, however, we draw a distinction between modern VMs and older concepts of virtualized resources. Virtual memory, for example, is fundamentally different from VMs in that it is intrinsically unstable and subject to *thrashing* [Gun95]. The isolation of modern VMs often helps to inhibit this kind of instability. (See Sect. 4)

The distinctive architectural feature which modern VMs have in common is some form of *polling* mechanism

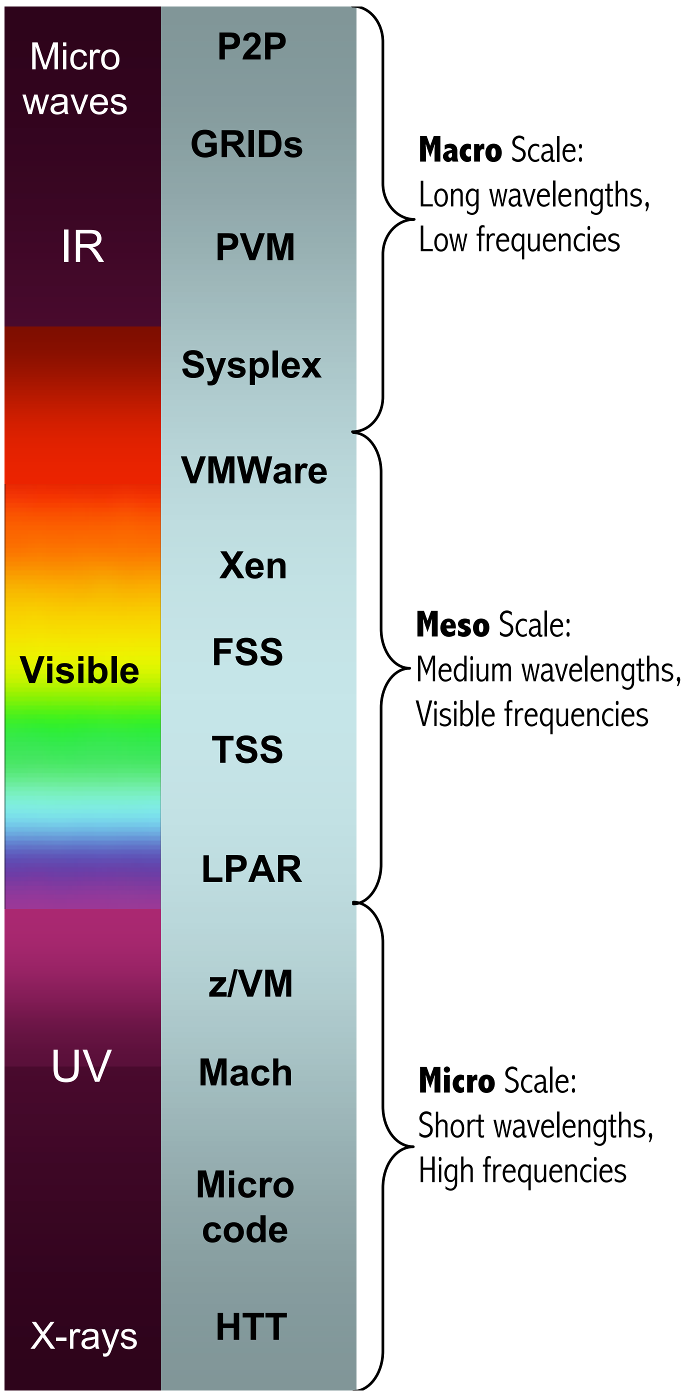

Micro waves
IR
Visible
UV
X-rays
P2P
GRIDs
PVM
Sysplex
VMWare
Xen
FSS
TSS
LPAR
z/VM
Mach
Micro code
HTT
Macro Scale: Long wavelengths, Low frequencies
Meso Scale: Medium wavelengths, Visible frequencies
Micro Scale: Short wavelengths, High frequencies

Figure 1: The continuous EM-spectrum compared with the discrete VM-spectrum. In both kinds of spectra, the relative location of each region is determined by their respective frequency scales (Table 1). For the VM-spectrum, the frequency is set by the VM polling rate. Like the invisible regions of the EM-spectrum, the micro and macro regions of the VM-spectrum are also less visible to standard performance management tools.

Table 1: VM-spectrum Scales

Spectral	Distance	Polling	
Region	Scale (m)	Period	Frequency
Macro	10^2 to 10^6	min to day	mHz to μHz
Meso	10^0 to 10^2	ms to min	kHz to mHz
Micro	10^{-6} to 10^{-3}	ns to μs	GHz to MHz

to accomplish resource sharing. Polling algorithms are intrinsically stable and fair; even round-robin polling exhibits intrinsic fairness. The potential performance penalty arises from the sequential nature of polling being worse than asynchronous communications. Based on this distinction, we can classify the variety of VM manifestations on a *discrete* spectrum, analogous to the *continuous* electromagnetic or EM-spectrum (Fig. 1).

2.1 Principal VM Regions

Just as the EM-spectrum can be grouped into the ultraviolet (UV), visible and infra-red (IR) regions, the VM-spectrum can be similarly grouped into the micro-VM, meso-VM, and macro-VM spectral regions. The relative position of each VM is defined by their respective polling rate or frequency scale in Table 1.

The so-called *visible* region on the EM-spectrum is an anthropocentric term. Certain snakes can see in the IR (detect heat) and bees can see in UV light. Similarly, only meso-VMs are "visible" to us via conventional performance management tools, in the sense of providing an immediate view of performance and thereby some level of potential control. Conversely, micro-VMs and macro-VMs tend to be invisible to those same tools, so they remain largely beyond our capacity management control.

From this standpoint, the distinction between VMs according to whether they are implemented in hardware or software, seems artificial; as artificial as the distinction between heat and light. Recognizing each as different manifestations of the same spectrum can lead to important insights. As we endeavor to show in the remainder of this paper, the VM-spectrum classification is more than mere whimsy.

2.2 Polling Rates and Frequency Scales

A key observation of this paper is that relative position of each VM sub-type on the VM-spectrum (Fig. 1) is determined by the rate at which polling is carried out by the underlying scheduling subsystem. To make this statement more quantitative, we refer to a case where the polling periods are well documented [Gun99]: meso-VM scheduling (see Sect. 4.1). The polling period T_p, for the scheduler to associate physical resource consumption with a each active OS instance (software VM), is once every 4000 ms or $T_p = 4$ s. The frequency is therefore $f = 1/T_p = 0.25$ cycles per second or 250 mHz.

We assume (because it is not documented) that the micro-VM polling period lies in the range of ns (the processor GHz clock frequency) to μs (MHz frequency). Macro-VMs can take minutes or days to detect active peer horizons. These frequencies are key VM performance determinants. For those who prefer to think in terms of size, a distance scale d (in meters) can be loosely related to the period T_p by $d = vT_p$ where v is the phase velocity of the communication signal. Typically, $v = c$; the speed of light. All these scales are summarized in Table 1.

3 MICRO-SCALE: HYPERTHREADS

We begin a detailed analysis of VMs starting with the highest frequency (smallest size) scale on the bottom of the VM-spectrum in Fig. 1; virtualization of physical processing resources. Intel, for example, refers to this form of processor virtualization as *hyper-threading technology* (HTT)) or *multithreading* (MT) on its Xeon™ and Pentium 4™ product lines. The rationale is to maximize throughput performance by utilizing idle cycles. Part of the current confusion over hyperthreading performance stems from two possible views of what HTT offers:

$(\mathbf{1}+\epsilon)$ **View:** This is the *hardware* perspective where ϵ is a small fractional quantity. Since there is a only *one* execution unit—which is often under-utilized—by simply duplicating a small number of registers, it becomes possible to have another thread ready to utilize any idle cycles. Intel quotes typical performance gains ranging from $\epsilon = 0.1$ to 0.3.

$(\mathbf{2}-\delta)$ **View:** This is the *software* perspective as seen by the OS and thus, performance management tools. An HTT-enabled processor presents itself to the OS as *two* logical or virtual processors (VPUs). The OS literally detects the number of VPUs (amongst other things) by interrogating Architecture State (AS) registers EAX and EBX on the chip (Fig. 2) using the APIC (Advanced Programmable Interrupt Controller) and CPUID IA-32 instructions. Ideally, one might expect $\delta \to 0$, but in reality $0 \ll \delta < 1$ so compute cycles appear to

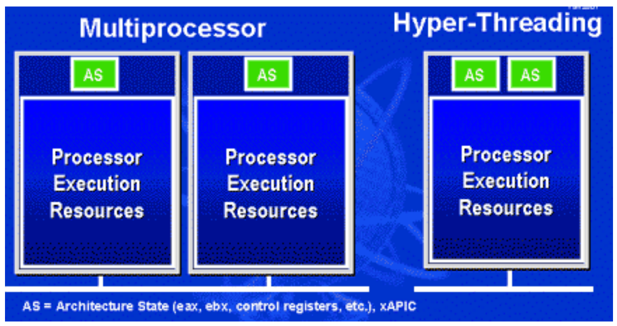

Multiprocessor
Hyper-Threading
AS
AS
AS
AS
Processor Execution Resources
Processor Execution Resources
Processor Execution Resources
AS = Architecture State (eax, ebx, control registers, etc.), xAPIC

Figure 2: Simple block diagram comparing a 2-way SMP (*left*) with an HTT-capable Intel processor (*right*). The two blocks labeled AS (Architectural State) are registers which present themselves to the OS as two VPUs. [Source: Intel Developer Forum]

be lost viz., the "Missing MIPS" problem referred to in the Introduction.

The relationship between these two views can be summarized simply as: $\delta = 1 - \epsilon$. In this paper, we take the $(2 - \delta)$ view because it best represents the source of confusion for many performance analysts and capacity planners. As we shall see, the starting point is closer to $\delta = \epsilon = 0.5$ for best case cpu-intensive workloads in both controlled benchmarks and some production workloads. More typically, since $\epsilon \ll 0.5$ it follows that $\delta \gg 0.5$ which results in a virtual capacity of $(2-\delta) \ll 1.5$ VPUs. This already tells us that part of the Missing MIPS problem is an illusion.

Hyperthreading can also be combined with *multicore* technology [KTJR05], where multiple physical CPUs are interconnected (like an SMP) on the same VLSI die. Sun Microsystems refers to this as a *chip multiprocessor* (CMP) and offers it with the UltraSPARC T1 processor comprising 8 cores with 4 threads per core for a total of 32-way VPUs. All the measurements presented in this paper, however, were made on Intel processors with HTT capability.

To further disambiguate physical CPUs from virtual VPUs in the subsequent discussion, we employ the simple mnemonic of a generic *polling system* [Gun05] in which multiple queues or buffers are multiplexed onto a common server or execution unit (Fig. 3). In the case of HTT processors there are just two queues corresponding to single-entry thread buffers viz., the AS state registers in Fig. 2. It is the state of these buffers that are monitored by the OS scheduler. Threads are taken off the run-queue and placed in the next empty AS buffer. On chip, each thread buffer is serviced by the single execution unit in some order e.g., round-robin for "fairness"

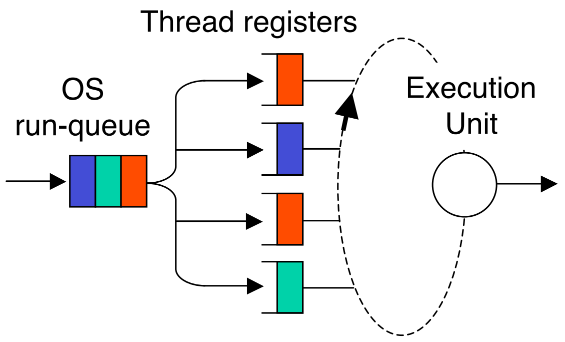


Figure 3: Simple polling model of a generic hyperthreaded processor with one execution unit servicing four thread registers or single-entry buffers (e.g. UltraSPARC T1). The AS registers in Fig. 2 correspond to two thread buffers (e.g. Intel Xeon).

(cf. Sect. 4), although the exact protocol may be quite complex and undocumented as part of micro-VM opacity. Hereafter, CPU shall refer to the execution unit or core processor, while VPU shall refer to the two AS registers or thread buffers. The polling model has not been widely recognized in this context and differs from the tandem-queue model presented in [DBK03]. The reader should note we are not suggesting that a constant polling delay (on the order of nanoseconds in Table 1) is responsible for the measured variation in HTT performance. That extension to the polling model is developed next.

3.1 Micro-VM Polling

Polling systems are not new e.g., token-ring networks use this principle, and are in common use for high-speed data network switches and routers. The performance characteristics of network polling systems are notoriously difficult to solve analytically [GCY03].

Careful measurements of hyperthreaded processors indicate that execution times can depend significantly on the type of applications being run (Sect. 3.2). While hyperthreading improves performance in many instances, some tests suggest that thread processing times are dependent on the load being borne by the thread-scheduler. Moreover, there are internal complexities such as how context switching is handled, whether L1 caches are shared as in Intel processors or independent L1 caches per core as in Sun's T1, so on. These invisible contributions to variability in processing times can lead to erroneous capacity forecasting without an appropriate performance model [See e.g., Fer05, Bra05, DBK03, Joh03].

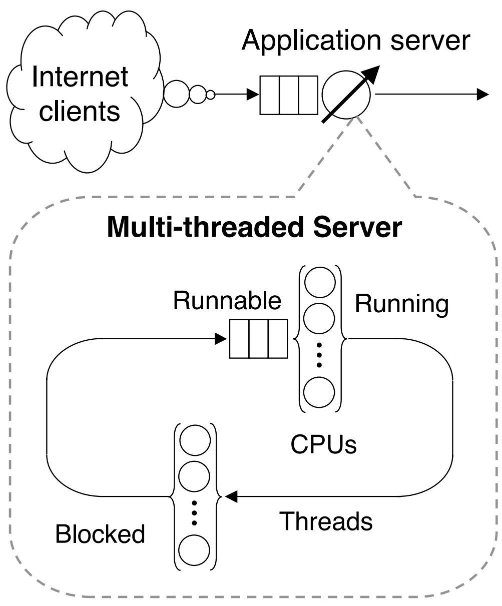


Figure 4: Extended PDQ model of a threaded web-application showing the load-dependent thread server.

An $M/G/1$ queue can be used to represent the performance of polling systems, but that requires rarely measured first and second moments of the service time [GCY03, Gun05]. Instead, we accommodate thread-state variability in Fig. 4 by aggregating the thread-buffers with the execution unit into separate load-dependent servers. Taken together with the OS run-queue this composite model more closely resembles an $M/M/m$ queue with a non-constant mean service time ($M/M/2$ for HTT).

3.2 Performance Analysis Examples

We apply the composite PDQ model of Sect. 3.1 to measurements of micro-VMs. Sect. 3.2.1 compares a multi-threaded test workload with HTT enabled and disabled. The data used in that portion of the analysis comes from [Joh03]. Sect. 3.2.3 is based on measurements of a production application with HTT enabled. The data used in that portion of the analysis comes from [Fer05].

3.2.1 Thread Execution Analysis

[Joh03] constructed a test program to consume all available CPU cycles by configuring the number of executing threads. The test platform comprised a dual-processor Compaq ML530 equipped with 2.4GHz Intel® Xeon™ processors running Microsoft Windows 2000™. A BIOS utility provided the ability to enable and disable HTT. Elapsed time was measured at 1 s resolution using the `time()` function. System and user processing time were measured using `GetProcessTimes()` at 100 ns resolution which included the activity of all process threads.

Table 2: Throughputs Calculated from [Joh03]

m	X_{off}	$X_{\text{off}}^{\text{PDQ}}$	X_{on}	$X_{\text{on}}^{\text{PDQ}}$
1	0.004739	0.0046989	0.004975	0.004698
2	0.009302	0.0093970	0.009434	0.009397
3	0.009288	0.0093970	0.014151	0.014096
4	0.009346	0.0093970	0.014493	0.018794
8	0.009346	0.0093970	0.014546	0.018794
16	0.009373	0.0093970	0.014599	0.018794

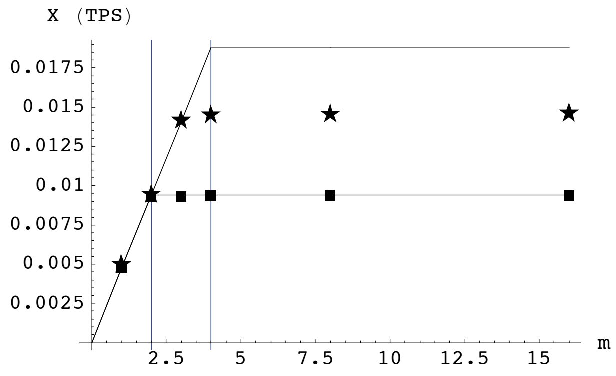


Figure 5: Predicted throughput (*solid curves*) and measurements for a test program exercising $m = 1, 2, \ldots, 16$ threads on a 2-way Intel platform with HTT disabled (*squares*) and enabled (*stars*). The "Missing MIPS" are quite apparent in the latter case.

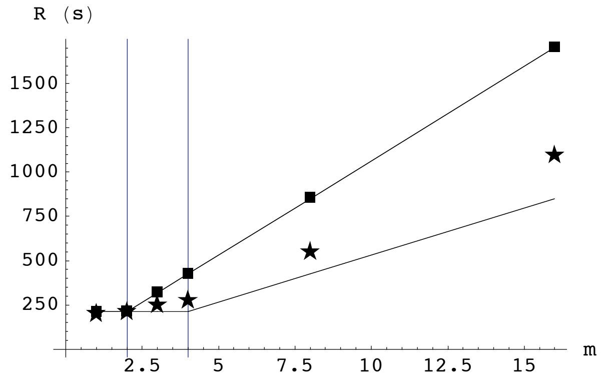
Throughputs in Table 2 were calculated from the data reported in [Joh03] using the definition:

$$X = m/T_m \;, \tag{1}$$

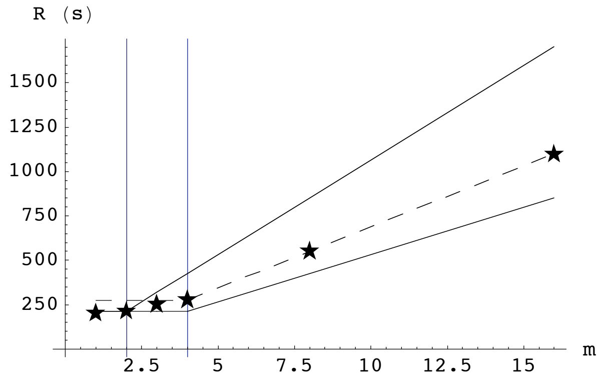
where m is the number of active threads and T_m the test program elapsed time. X_{on} denotes HTT enabled and conversely for X_{off}. Fig. 5 compares these calculated throughputs with those predicted by a polling model in PDQ [Gun05]. With HTT disabled (lower curve), measurement and prediction are almost identical and a knee occurs at $m = 2$ (or 2 CPUs). With HTT enabled (4 VPUs), the knee occurs earlier than $m = 4$ and prevents predicted throughput from being achieved. This is the "Missing MIPS" problem referred to in Sect. 1. The explanation is provided by analyzing the runtimes.

3.2.2 Missing MIPS Explained

Fig. 6 compares runtime data with polling model predictions. With HTT disabled (2 CPUs), the data fall on the upper curve with the expected knee occurring at $m = 2$. With HTT enabled (4 VPUs), the data points lie *above* the predicted lower curve by about 30%. The predicted curves in Fig. 6(a) assume a constant mean service time per thread S_0. For a processor-intensive workload with a finite number of threads active during each measurement, the predicted runtime curve should increase *linearly* above saturation ($m = 2$ or 4) because it is a *closed* queueing system [cf. DBK03]. But these data are *super-linear* relative to the lower curve.



(a) Predicted elapsed times and measurements for Fig. 5. Without HTT the data (*squares*) match PDQ predictions (*upper curve*) very closely with the knee occurring at $m = 2$ (i.e., 2 CPUs), but the expected improvement with HTT enabled (*lower curve*) is not fully realized for $m \geq 4$ (*stars*).



(b) The longer elapsed times under HTT can be accounted for by increasing the service time in the PDQ model by 20% (*dashed line*). The service times are constant but a transition to a longer service time begins at $m = 3$.

Figure 6: Measured runtimes corresponding to Fig. 5.

We can equate runtime measured by the OS to residence time R at the VPU. In Fig. 6(a) the residence time:

$$R_0(m \leq 4) = S_0 \ , \tag{2}$$

is simply the constant service time S_0 at the foot of the "hockey stick", signifying a processor is always available to service threads and no waiting time is incurred. When $m > 4$, however, all processors become saturated and threads begin to queue in the buffers of Fig. 3. For a saturated closed queueing model, $X(m) = 1/S_0$ and the residence time above $m = 4$ is:

$$R_0(m > 4) = mS_0 - Z \ , \tag{3}$$

which is linear rising in m (cf. Fig. 6). For these data, the thinktime $Z = 0$ so that processors are 100% busy during the tests. Previous authors have speculated that increased wait time (mS_0) may be responsible for the observed increase in R [See e.g., Joh03, DBK03, Fer05], but with a fixed number of threads, how can the thread-wait time increase *super-linearly*? The answer is, S_0 has increased to a new value $S_b > S_0$. In other words, (3) has now become:

$$R_b(m > 4) = mS_b \ , \tag{4}$$

such that R_b is still linear rising (dashed curve in Fig. 6(b)), but at a increased angle relative to R_0. Of the 30% increase in R_b, PDQ reveals that 20% is due to a sudden increase in the thread-*service* time. It seems reasonable to conclude that this increase $(S_b - S_0)$ is associated with the extra time needed for internal state management, as described in Sect. 3.1, when the number of thread requests exceeds the number of VPUs (empty buffers in Fig. 3).

Using the terminology of Fig. 2, the dual-core Compaq ML530 has two independent sets of AS buffers denoted 1a0, 1a1 belonging to core 1, and 2a0, 2a1 on core 2. In Fig. 6(b), the elapsed times start out on the lower hockey stick because threads are likely being assigned to available VPUs (empty thread buffers) in the order 1a0, 2a0 i.e., one thread per core. The third thread has to be assigned to a core that is already busy (probably 1a1). Notice that the elapsed time for $m = 3$ in Fig. 6(b) appears to "lift off" the lower hockey stick, reflecting the extra time needed for internal management of the micro-VM registers and caches. The fourth thread is then assigned to buffer 2a1 on already busy core 2, whereupon the transition to the upper hockey stick (dashed curve) in complete. The increase in service times is reflected in OS measurements as prolonged execution times.

The foregoing analysis was based on the controlled cpu-intensive workload in [Joh03]. IO-intensive workloads would likely show a different elapsed time profile but our expectation is that that they too can be analyzed using the same or a similar PDQ model (See e.g., Sect. 4.2.2).

3.2.3 Windows 2000 Production Server

Missing MIPS are also seen in production workloads. [Fer05] discusses performance measurements analyzed with BMC Perform/Predict®. The production system is a dual-core Dell 2650 platform with HTT enabled and running Microsoft Windows 2000®. The puzzle is to account for web application "CPU-wait" in spite of available processor capacity [cf. Bra05]. Specifically, high-priority CPU busy never exceeds 85% during peak demand, and Perform/Predict also indicates that CPU time is the major component of response time, rather then disk IO or memory accesses.

Drill-down analysis shows that the system was configured with 4 VPUs, since HTT was enabled, and the available processing capacity was therefore reported by Perform/Predict as 400%. Of this, $323.26 \pm 5\%$ was being utilized by the web application, with an average of 80.82% per VPU. From Sect. 3, we can write $(2-\delta) = 1.62$ VPUs per core or $\epsilon = 0.62$. Even allowing for 5% measurement error, this corresponds to excellent HTT efficiency so, we can assume that each physical execution unit is actually running at 100% busy with no idle cycles remaining. In other words, there are no more processor cycles available to do real work.

The paradox is resolved by noting that each processor is being reported as 81% busy (up to 85% at peak) by the performance management software, but that utilization is calculated incorrectly on the basis of VPUs. From Sect. 3 we know that the implied under-utilization is a misdirection. On the other hand, because of the aforementioned micro-VM opacity, performance management tools have nothing else to go on.

3.3 Recommendations

Micro-VMs, in the form of VPUs, should not be regarded on the same footing as physical CPUs. They are more properly regarded as sophisticated polling systems, polled at rates in the GHz to kHz range, with the number of VPUs corresponding to the number of single-entry thread buffers. Internal state management in these micro-VMs introduces an intrinsic and often variable overhead.

The preceding performance analysis shows that the perceived $(2-\delta)$ missing MIPS problem is really an illusion due to not recognizing that the number of VPUs is actually $1+\epsilon$ where $\epsilon = 1-\delta$. The sudden prolongation of elapsed times can be explained by the prompt increase in service time required for internal micro-VM management when the VPU buffers are fully occupied. This overhead is invisible to the OS and cannot be tuned; only disabled on Intel CMPs.

The value of ϵ is also likely to vary between CMP releases from the *same* vendor, as well as across CMPs from *different* vendors. Because of the aforementioned lack of visibility on the VM-spectrum, qualifying micro-VMs (possibly using some of the methods in Sect. 3.2) should become a part of your capacity planning practice during hardware procurement and software acceptance testing.

4 MESO-SCALE: Hypervisors

VMWare® and Xen (`www.xensource.com`) are two examples of software-based VMs which offer a useful of array of new capabilities such as server consolidation, co-located hosting, distributed web services, isolation, secure computing platforms and application mobility [Zei04]. If the VMs are likened to musicians in an orchestra, the conductor is called the *hypervisor* or *virtual machine monitor* (VMM).

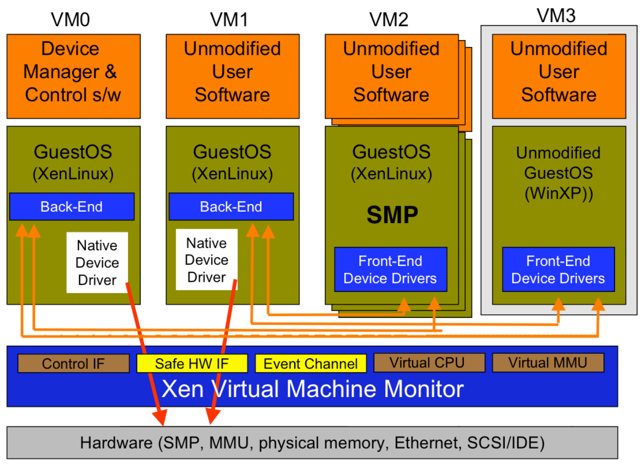


Figure 7: Organization of Xen 3.0 hypervisor supporting Linux, Linux SMP, and Windows XP meso-VMs.

The partitioning resources in a physical machine to support the concurrent execution of multiple VMs poses several challenges (Fig. 7). First, the VMs must be truly isolated from one another. It is unacceptable for the execution of one VM to adversely affect the performance of another. This is particularly true when virtual machines are owned by mutually untrusting users. Second, it is necessary to support a variety of different OS instances to accommodate the heterogeneity of popular applications. Third, and most importantly from a capacity planning standpoint, the performance overhead introduced by VMMs should be small.

Whereas time-share scheduling (TSS) in Fig. 8 provides each user with the illusion that she is the only user of the physical processor, fare-share scheduling (FSS) Fig. 9 provides each user (or group of users) with the illusion that she possesses an entire platform—a virtual machine—whose performance is scaled according to her resource entitlement. Entitlement ($\mathcal{E}$ in Table 4) is awarded by the system administrator through the allocation of *shares* (like owning shares in a corporation).

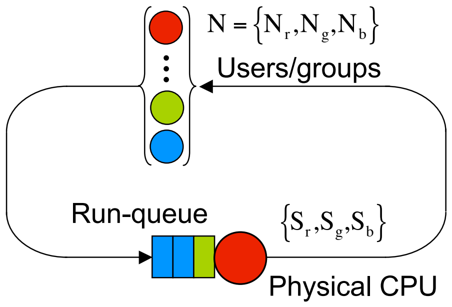


Figure 8: Time-share scheduler model.

Xen (Fig. 7) uses a form of FSS called *Borrowed Virtual Time* (BVT) as the default scheduler; other options are also available e.g. real-time [BDF+03]. BVT provides proportional FSS for processor scheduling based on weights. Each runnable domain receives a share of the processor in proportion to its weight. A single processor VMWare guest OS gets 1000 shares by default [VMw05]. The impact of share allocation on performance is discussed in Sect. 4.2.

4.1 Meso-VM Polling

The most important attribute of FSS for this paper is that it employs a polling mechanism to govern resource sharing at runtime. Fig. 8 shows a PDQ model of TSS with three different process classes (N_r, N_g, N_b) each of which is in one of three possible states: runnable (waiting in the run-queue), running (on a processor) or suspended (in the upper part of the diagram). If a request has not completed execution when the time-quantum expires (e.g., 10 ms or 50 ms in VMWare) it is returned to the tail of the run-queue. Processes waiting for other resources (e.g., I/O requests) are suspended.

The PDQ model of FSS in Fig. 9 depicts each red, green and blue user-process of the TSS model having been allocated their own VM whose service time is scaled by

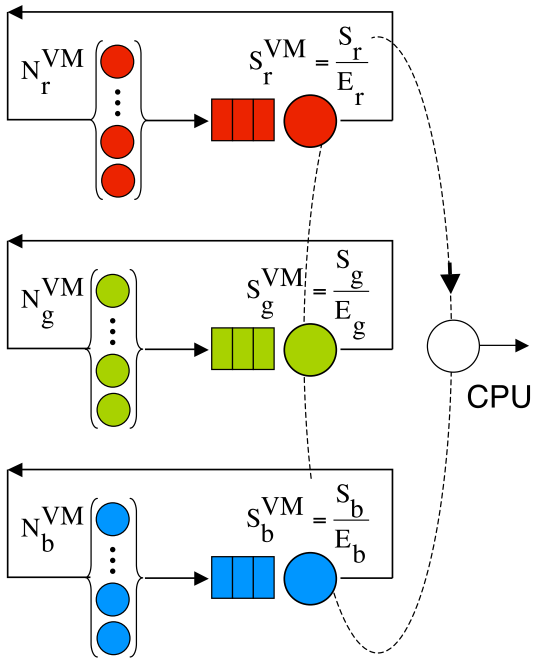

N_r^{VM}
$S_r^{VM} = \frac{S_r}{E_r}$
N_g^{VM}
$S_g^{VM} = \frac{S_g}{E_g}$
N_b^{VM}
$S_b^{VM} = \frac{S_b}{E_b}$
CPU

Figure 9: Fair-share scheduler polling model of a meso-VM like Fig. 7. The hardware platform has the same logical association to the guest VMs as the physical CPU does to the VPUs in Fig. 3.

their respective share entitlements $\mathcal{E}_g$ running under the supervision of the VMM. Consequently, the actual service time S_g for guest instance g becomes the *virtual* service time

$$S_g^{VM} = \frac{S_g}{\mathcal{E}_g}, \tag{5}$$

as indicated by the runtimes in Table 4. Each guest virtual server (VM) is polled by the VMM on behalf of the processors in the physical platform. The polling rate operates at a frequency of around 250 mHz (Table 1). Note the similarity with Fig. 3.

FSS [KL88] introduces a scheduling superstructure on top of conventional TSS to connect processes with users and their resource entitlements as represented in the following (highly simplified) pseudocode:

VM Share Scheduling: Polls every 4000 ms ($f = 250$ mHz) to compare physical processor usage per user entitlement (Fig. 9).

```
for(i = 0; i < USERS; i++) {
    usage[i] *= decayUsage;
    usage[i] += cost[i];
    cost[i] = 0;
}
```

VM Priority Adjustment: Polls every 1000 ms ($f = 1$ Hz) and decays internal FSS process priority values (Fig. 9).

```
priDecay = Real number in the range [0..1];
for(k = 0; k < PROCS; k++) {
    sharepri[k] *= priDecay;
}
priDecay = a * p_nice[k] + b;
```

Time Share Scheduling: Polls every physical processor tick ($f = 100$ MHz) to adjust process priorities (Fig. 8).

```
for(i=0; i<USERS; i++) {
    sharepri[i] += usage[i] * p_active[i];
}
```

Process-level polling is essentially the same as standard TSS, while VM-share polling controls process-level capacity consumption.

4.2 Performance Analysis Examples

In this section we analyze examples of meso-VM performance. The data used in Sects. 4.2.1 and 4.2.3 are based on internal benchmarks conducted by VMWare engineers on an ESX Server and published in [VMw05]. Although these data are extremely useful, one has to remain mindful that such internally conducted benchmarks are often selected so as to avoid illuminating the less than favorable performance aspects of a vendor's product. The data for the J2EE/WebLogic application in Sect. 4.2.2, on the other hand, come from tests that were conducted by a client under my supervision. These data are not skewed by any commercial considerations, but details concerning the type of application had to be kept confidential.

4.2.1 VMWare Share Allocation Analysis

VMware ESX Server 2.5.1 provides a middleware layer that enables users to create multiple independent VMs on the same physical server. Benchmark experiments employed processor-intensive workloads which consumed 100 percent of available processing resources. A single application called 164.gzip from the SPEC CPU2000 benchmark suite (www.spec.org), was used as the workload. The SPEC version of the GZIP data compression code does not perform any file I/O other than reading the input, and all compression/decompression is performed in memory. More importantly, the workload runs in user-space and therefore induces very little overhead between

Table 3: ESX 2 Benchmark Measurements [VMw05]

Active VMs		Shares per VM		Runtime (s)	
VM$_{hi}$	VM$_{lo}$	\$$_{hi}$	\$$_{lo}$	R$_{hi}$	R$_{lo}$
1	7	2000	1000	1296	2352
1	7	2333	1000	1157	2357
1	7	2000	857	1153	2350
2	6	2000	1000	1470	2363
2	6	3000	1000	1159	2359
3	5	5000	1000	1159	2360

Table 4: PDQ Model Predictions for Table 3

Active VMs		Entitlements		Runtime (s)	
VM$_{hi}$	VM$_{lo}$	$\mathcal{E}_{hi}$	$\mathcal{E}_{lo}$	R$_{hi}$	R$_{lo}$
1	7	0.2222	0.7778	1296.00	2592.00
1	7	0.2500	0.7500	1152.12	2687.90
1	7	0.2500	0.7500	1151.86	2688.11
2	6	0.4000	0.6000	1440.00	2880.00
2	6	0.5000	0.5000	1152.00	3456.00
3	5	0.7500	0.2500	1152.00	5760.00

guest OS kernel and the VMM. See Sect. 4.3 for more on these limitations.

With the in mind, this VMWare study is nonetheless useful from the standpoint of quantifying the potential impact of different share allocation choices on meso-VM performance. Such data are otherwise often difficult to come by. The benchmark tests were conducted on an 4-way HP ProLiant DL580 server employing 2.2GHz Intel Xeon® processors with HTT disabled. Although the SPEC gzip benchmark does not represent a very realistic workload, we note that [Bra05] has reported performance anomalies for a production tar/gzip file-compression application running on a system with IBM z/VM® as the hypervisor.

Table 3 summarizes the benchmark results with different share allocations for the 8 VMs where between 1 and 3 VMS are executed at high priority i.e., a larger proportion of the share pool. Table 4 summarizes the corresponding performance prediction using a PDQ model based on Fig. 9. The runtimes (R_{hi}) for the high-priority VMs are in very close agreement with the measurements. The increasing divergence between the predicted and measured values of R_{lo} is easily explained by noting that the tests allowed each instance of the SPEC gzip code to run to completion, whereas PDQ assumes the tests are run in steady state. In the tests, when a high-priority VM completed those processor cycles became available to the high-priority VMs, allowing then to complete in near constant time.

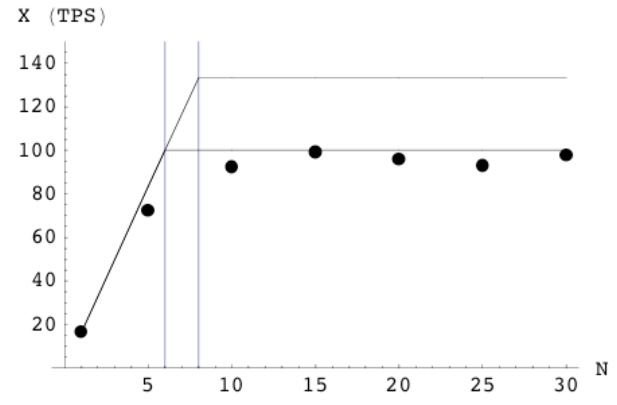


Figure 10: Predicted throughput (*solid curves*) and measurements (*dots*) on a WebLogic J2EE production application. The PDQ model exposes the missing MIPS.

4.2.2 J2EE WebLogic Production Application

Missing MIPS are also observed in production meso-VM applications. Fig. 10 shows transaction per second (TPS) measurements for a J2EE/WebLogic® application accessing a Sybase® database. Measurements were conducted on an isolated Dell PowerEdge 1750 server with dual 3.06 GHz Xeon processors. HTT was enabled under Windows Server 2003 Enterprise Edition®. LoadRunner® generated a controlled workload with $N = 1, 2, \ldots, 30$ virtual users and the maximum achieved throughput was 100 TPS. JXInsight™ provided traces from which PDQ service times were extracted as well as revealing that Sybase was not the bottleneck.

Each WebLogic server (or VM) has a single execute queue supported by 25 threads. If all 25 WebLogic threads could do real work, PDQ predicts a maximum application throughput of 415 TPS starting at $N = 25$ vusers. In fact, Fig. 10 shows that we only observe 100 TPS or about one quarter of the expected throughput. The explanation is as follows. With HTT enabled, we have $2\text{-way} \times 2 = 4$ VPUs virtual capacity from Sect. 3. The WebLogic architecture involves *listen threads* (not to be confused with the TCP/IP listen queue) that gate work onto the execute queue. WebLogic assigns 2 listen threads per processor which, from the viewpoint of WebLogic/Windows OS on this HTT-enabled platform, translates to initiating $4 \text{ VPUs} \times 2 = 8$ listen threads. The knee in the throughput profile is therefore more properly expected at $N = 8$ vusers, corresponding to a system throughput of 133 TPS (upper curve in Fig. 10). The observed throughput, however, exhibits a premature knee at $N = 6$ or about 75% of the assumed VPU capacity (cf. Fig. 5). From Sect. 3, we recognize that there are only $2\text{-way} \times 1.5 = 3$ VPUs or 6 active WebLogic listen-threads running concurrently, hence the knee at $N = 6$ and the observed maximum throughput of only 100 TPs. Despite having a pool of 25 threads available to service the WebLogic execute queue, these six listen threads are the performance limiter. With HTT disabled, the expected maximum throughput would be only 67 TPS at $N = 4$ vusers i.e., $2\text{-way} \times 2 = 4$ listen threads.

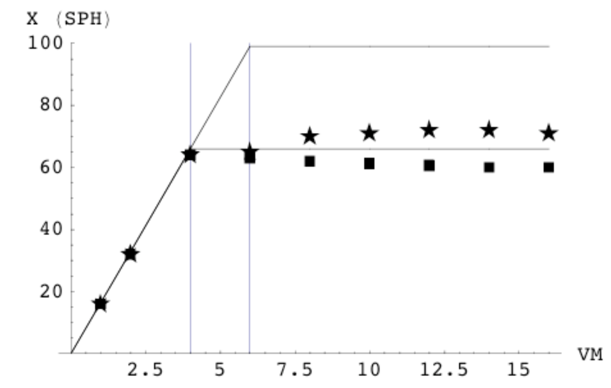


Figure 11: VMWare throughput measured in scripts/hr as function of active guests with HTT disabled (*squares*) and enabled (*stars*).

4.2.3 VMWare Scalability Analysis

In the preceding example, a meso-VM is running on a micro-VM such that the results might be confounded by possible interactions between VM levels. To separate these effects out, Fig. 11 shows measured VMWare throughput as a function of active VMs together with with HTT separately enabled and disabled.

With HTT disabled, the 4-way ProLiant DL580 exhibits no missing MIPS. Besides a moderate decline of about 5 SPH (scripts per hour), possibly due to increasing VMM overhead, the throughput ceiling of 66 SPH commences at 4 VMs or guests. With HTT enabled, the 4-way HP ProLiant DL580 server presents $4\text{-way} \times 2 = 8$ VPUs to VMWare and should therefore exhibit a knee at 8 VMs. Recalling Sect. 3 however, it is more realistic to expect the actual virtual capacity to be closer to $4\text{-way} \times 1.5 = 6$ VPUs. Fig. 11 reveals that even this expectation is not met when the micro-VM and meso-VM levels interacting. Presumably the loss in throughput is due to overheads in VMWare in this case. Without additional instrumentation, this level of detail remains opaque.

4.3 Recommendations

Meso-VMs are also implemented as polling systems operating at rates in the kHz to mHz range. Proportional shares are used to create software VPUs in which the service rate is scaled by share-based entitlements according eqn.(5). Proper share allocation can be critical for capacity management [Gun99] and controlled measurements like those in Sect. 4.2.2 should be considered essential for proper capacity planning.

All meso-VM measurements should be made in *steady state* i.e., where the difference between the average number of requests and the average number of completions becomes vanishingly small [Gun05]. This would exclude potentially misleading side-effects, like early completions benefiting late completions in Table 3.

In contrast to the limited perspective offered by the SPEC gzip workload in Sect. 4.2.1, the interested reader can find a more encompassing set of benchmark data in [BDF$^+$03]. These data show that both Xen and VMWare may exhibit significant performance degradation relative to CPU-bound workloads due to VM overhead. For example, the SPEC WEB99 benchmark shows 70% degradation relative to SPEC CPU2000, while an OLTP workload shows as much as 90% relative degradation. Unfortunately, these results were not measured relative to share allocations, which was the purpose in Sects. 4.2.1 and 4.2.3.

Moreover, exercising more realistic workloads in no way detracts from the usefulness of our PDQ models because the service time S_g in (5) is the sum of user-time and kernel-time, and both contributions are measured as part of the system response to the workload. If this were not true, then the analysis of the production system in Sect. 4.2.2, which involves network interactions with the Sybase database, could not be validated.

5 MACRO-SCALE: GRIDS and P2P

In this section we consider virtualization associated with large-scale macro-VMs such as GRIDs and peer-to-peer (P2P) hypernet networks. The latter include Gnutella (Fig. 12), Napster, Freenet, Limewire, Kazaa, `SETI@Home`, BitTorrent, Skype (Fig. 13), instant messaging, WiFi, PDAs and even cellphones. They have progressed from simple one-off file transfers to a scalable means for distribution of applications such as games, movies, and even operating systems.

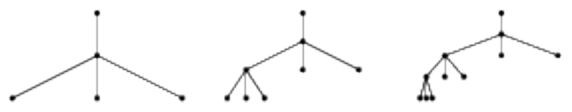


Figure 12: Cayley trees with degree-4 vertices similar to those used in P2P networks like Gnutella and Napster.

Although P2P networks and GRIDs share the common focus of harnessing resources across multiple administrative domains, they can be distinguished as follows. GRIDs support a variety of applications with a focus on providing infrastructure with quality-of-service to moderate-sized, homogeneous, and partially trusted communities [Fos05]. P2P supports intermittent participation in vertically integrated applications for much larger communities of untrusted, anonymous individuals. P2P systems provide protocols for sharing and exchanging data among nodes. The network architecture tends to be more decentralized, and dynamics requiring resource discovery.

GRID computing has focused on scientific and engineering applications where it attempts to provide diverse resources that interoperate [GTN$^+$05]. The concept behind the GRID is analogous to the electrical power grid. When you throw the switch, you expect the light to come on. GRID computing is most often discussed within the context of scientific and engineering applications because they are generally very CPU-intensive. ASCI BlueMountain, part of ASCI-Grid with 6144 processors, employs FSS job scheduling [KC03]. See [Str05] for an overview of GRIDs in the commercial enterprise.

These technologies are not mutually exclusive. P2P technologies could be used to implement GRID systems that avoid or alleviate performance bottlenecks [TT04]. Although these technologies are still rapidly evolving, applications are becoming more robust (it's not just about music files anymore), so capacity planners should prepare themselves for the occasion when these macro-VMs connect into your data center.

5.1 Macro-VM Polling

Polling protocols are employed by macro-VMs in at least two ways: maintaining connectivity between peers, and security on the network. Each type of polling protocol has important ramifications for network performance and capacity. Although generally more nebulous and system specific than micro-VM or meso-VM polling mechanisms, the particular case of wireless networks (see IEEE 802.11 standard) provides an illustrative example of their potential performance impact.

When carrying both voice and data, VoIP packets require contentionless periods in the transmission protocol, whereas data packets can tolerate contention (simple retry). Wireless access points poll, regardless of whether data is available for transmission or not. When the number of stations in the service set grows, the polling overhead is known to become large. Without some kind of service differentiation, performance degrades. One enhancement that has been considered to increase network capacity is a polling list where idle nodes are dynamically deleted or active ones are added. This helps to increase the number of contentionless periods thereby improving WLAN capacity by about 20%.

Polling to maintain P2P network security is employed in the sense of collecting opinions or votes. Providing security for distributed content sharing in P2P networks is an important challenge due to vulnerabilities in many protocols for sharing the "reputations" of peers. Certain polling protocols are subject to attacks which can alter the results of any voting procedure. Securing macro-VM networks has capacity planning implications.

5.2 Performance Analysis Examples

The goal of macro-VMs is to enable scalable virtual organizations to provide a set of well-defined services. Key to performance is the network topology and its associated bandwidth. To assess the scalability of network bandwidth, this section draws on performance bounding techniques described in [Gun05, Chap. 5]. Since very few reliable measurements exist for these large-scale systems, the data in this section are purely theoretical and due to the author.

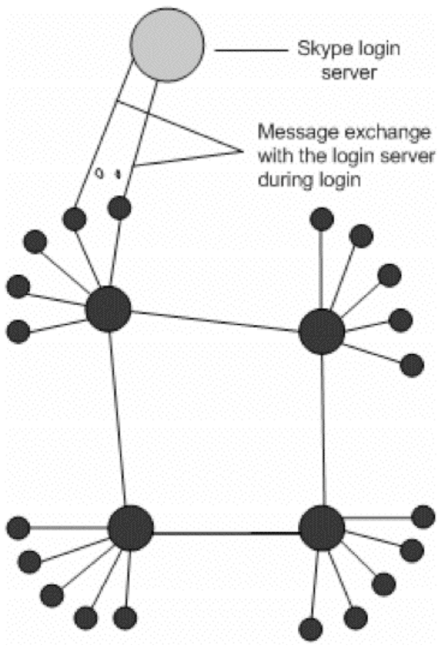


Figure 13: Skype hypernet network showing both peers and *super peers* (*large black dots*).

Table 5: P2P hypernet topologies ranked by maximal relative bandwidth (BW), showing connections per peer (C/N), average number of network hops (H), and the number of supported peers (N) in millions.

Hypernet Topology	C/N	H	$N \times 10^6$	BW
20-Cube	20	10	2.1	100
10-Torus	20	11	2.1	93
20-Cayley	20	6	2.8	16
8-Cayley (Napster)	8	8	1.1	13
4-Cayley (Gnutella)	4	13	1.1	8

5.2.1 Bandwidth Scalability Analysis

The main results are summarized in Table 5 which shows each of the topologies ranked by their relative bandwidth. The 20-dimensional hypercube outranks all other contenders on the basis of query throughput. For an horizon containing 2 million peers, each servant must maintain 20 open connections, on average. This is well within the capacity limits of most TCP/IP implementations. The 10-dimensional hypertorus is comparable to the 20-hypercube in bandwidth up to an horizon of 1 million peers but falls off by almost 10% at 2 million peers.

The 20-valent Cayley tree is included since the number of connections per peer is the same as that for the 20-cube and the 10-torus. An horizon of 6 hops was used for comparison because the peer population is only 144,801 nodes at 5 hops. Similarly for 8-Cayley, a 9 hop horizon would contain 7.7 million peers. These large increments are a direct consequence of the high vertex degree per node. The 4-Cayley (early Gnutella network in Fig. 12) and 8-Cayley (Napster network) show relatively poor scalability at 1 million peers [Rit02]. Even doubling the number of connections per peer produces slightly better than 50% improvement in throughput.

Because bandwidth in these topologies grows in proportion to added nodes or peers (Fig. 14), no throughput ceiling of the type appearing in Figs. 5, 10 and 11 is observed. *BitTorrent* is a P2P file-sharing protocol which effectively implements higher-order topologies dynamically in software. Every client downloading a file from the network usually donates part of its own bandwidth, making it much faster than earlier P2P technologies like Gnutella or Kazaa.

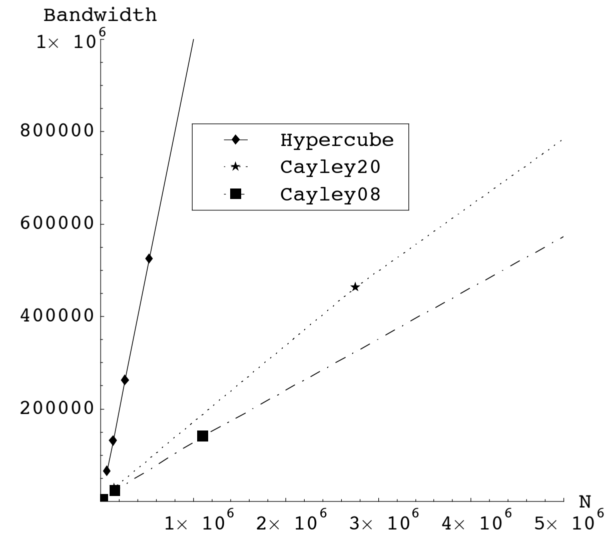


Figure 14: Predicted bandwidth as a function of peers (N) for different hypernet topologies in Table 5.

5.2.2 Remote Polling Rates

Though BitTorrent is a good protocol for broadband, it is less effective for dial-up, where dropped connections are common. On the other hand, many HTTP servers drop connections over several hours, while many torrents exist long enough to complete a multi-day download often required for large files. An uploading client is flagged as snubbed if the downloading client has not received any data from it in over 60 seconds.

Some BitTorrent clients also report the *share ratio*, a number relating the amount of data uploaded to the amount downloaded. A share ratio of 1.0 means that a user has uploaded as much data as they have downloaded. Some networks, for example, prevent access to new torrents for the first 24–48 hours (i.e., up to 1.7×10^5 seconds in Table 1) that the torrent is active to people with overall ratios of less than 1.0 and a certain amount of data uploaded.

5.3 Recommendations

It is more difficult to make many practical recommendations for meso-VMs because they are still emerging technologies. Sun Microsystems CEO Jonathan Schwartz, a major proponent of enterprise GRID computing, recently stated: *"Behind the corporate firewall, the transformation toward multi-tenant grids has been slower. Frankly, it's been tough to convince the largest enterprises that a public grid represents an attractive future. But things are changing."* Nonetheless, we suggest that GRIDs and P2P are more properly regarded as a legitimate region of the VM-spectrum (Fig. 1).

Two important points for capacity planners. First, adding nodes in macro-VMs adds bandwidth, so the throughput ceilings seen in Figs. 5, 11 and 10 are not expected to appear. Second, macro-VMs are mostly invisible to standard performance management tools, but some of the same performance analysis techniques discussed in Sects. 3.3 and 4.3 should be applicable as these technologies begin to connect to your data center.

6 CONCLUSION

Modern computing systems that abstract virtual resources from physical resources have surpassed the measurement paradigms of most performance management tools, thus they remain largely opaque to the performance analyst and capacity planner. As noted so prosaically in [Fer05], dealing with virtualization is like being adrift in a vast "sea" of unknowns where one hopes to spot some occasional "islands" of familiarity.

We have attempted to improve familiarity by introducing a spectral classification for VMs with proportional polling schedulers. This immediately distinguishes them from other forms of virtualization e.g., virtual memory. A poll-based classification also means that subcategories of VM architectures (micro, meso, macro) can be defined in terms of polling frequency; analogous to the UV, visible, and IR frequencies of the EM-spectrum.

Polling rates also set the operational scale of VMs and we have shown that, just like the visible region of the EM-spectrum, meso-VMs are the most "visible" to performance management control. While not perfect, the capacity planner has the immediate and measurable option of tuning performance and setting capacity limits by adjusting shares awarded to each VM. Guidance when allocating shares for performance needs to be improved, and performance models of the type developed here can play a significant role.

The macro-VM spectral region is largely invisible to conventional performance management tools and will likely remain that way for the foreseeable future. The prognosis is reminiscent of the reduced performance management capabilities for current web-based applications.

Curiously, the news may be somewhat better in the micro-VM region. Multithreaded applications like Java (Sect. 4.2.2) are very well-suited to HTT-type hyper-threading. [Sut05] has pointed out that the onus is now on application developers to understand or re-learn the subtleties of concurrent programming, but this time applied to micro-scale CMPs. Concurrency was already extremely important for meso-scale SMP applications. It may be *vital* for micro-scale CMP applications. Perhaps the VM-spectrum classification presented here can provide the beginnings of a useful framework for future discussions about virtualized systems.

7 ACKNOWLEDGMENTS

I am pleased to acknowledge G. Dawe, J. Rybicki, and A. Sliwkowski at RSA Security for the painstaking performance measurements which led to the development of some of the PDQ models used in this paper. J. Brady, G. Fernando, M. Friedman, S. Jenkin, S. Johnson, J. Peek, and G. Rogers provided many valuable comments on earlier drafts of this paper.

References

[BDF+03] Paul T. Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. "Xen and the art of virtualization". In *SOSP (ACM Symposium on Operating Systems Principles)*, pages 164–177, 2003.

[Bra05] J. Brady. Virtualization and CPU wait times in a Linux guest environment. *J. Comp. Resource Mgmt.*, 116:3–8, Fall 2005.

[DBK03] Y. Ding, E. Bolker, and A. Kumar. "Performance implications of hyper-threading". In *Proc. CMG*, pages 21–29, Dallas, TX, December 7–12 2003.

[Fer05] G. Fernando. "To V or not to V: A practical guide to virtualization". In *Proc. CMG*, pages 103–116, Orlando, FL, December 7–12 2005.

[Fos05] I. Foster. "Service-oriented science". *Science*, 308:814–817, May 6 2005.

[Fri03] M. Friedman. "Hyperthreading: Two for the price of one?". *CMG MeasureIT*, 1.0, March 1 2003.

[GCY03] N. Gunther, K. Christensen, and K. Yoshigoe. "Characterization of the burst stabilization protocol for the RR/CICQ switch. In *IEEE Conf. on Local Computer Networks*, Bonn, Germany, October 20-24 2003.

[GTN+05] L. Gilbert, J. Tseng, R. Newman, S. Iqbal, R. Pepper, O. Celebioglu, J. Hsieh, and M. Cobban. "Performance implications of virtualization and hyper-threading on high energy physics applications in a grid environment". In *Proc. 9th IEEE International Parallel and Distributed Processing Symposium*, page 32a, Denver, CO, April 3–8 2005.

[Gun95] N. Gunther. "Performance modeling of large transients in computer systems. In *Proc. CMG*, pages 338–347, Nashville, TN, Dec 4-8 1995.

[Gun99] N. Gunther. "Capacity planning for Solaris SRM: All I ever wanted was my unfair advantage (And why you cant get it!)". In *Proc. CMG*, pages 194–205, Reno, Nevada, December 1999.

[Gun05] N. Gunther. *Analyzing Computer System Performance Using Perl::PDQ*. Springer-Verlag, 2005.

[Joh03] S. Johnson. "Measuring CPU time from hyper-threading enabled Intel processors". In *Proc. CMG*, pages 369–378, Dallas, TX, December 7–12 2003.

[KC03] S. D. Kleban and S. H. Clearwater. "Hierarchical dynamics, interarrival times and performance". In *Proc. SuperComputer2003*, pages 28–34, Phoenix, AZ, Nov 15–21 2003.

[KL88] J. Kay and P. Lauder. "A fair share scheduler". *Comm. ACM.*, 31:44–55, 1988.

[KTJR05] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan. "Heterogeneous chip multiprocessors". *IEEE Computer*, 38(11):32–38, November 2005.

[Rit02] J. Ritter. "Why Gnutella can't scale. No, really.". `www.darkridge.com/~jpr5/doc/gnutella.html`, 2002.

[Sin04] A. Singh. "An introduction to virtualization". `www.kernelthread.com/publications/virtualization/`, February 5 2004.

[Str05] P. Strong. "Enterprise grid computing". *ACM Queue*, 3:50–59, July/August 2005.

[Sut05] H. Sutter. "The free lunch is over: A fundamental turn toward concurrency in software". *Dr. Dobb's Journal*, 30(3), March 2005.

[TT04] D. Talia and P. Trunfio. "A P2P grid services-based protocol: Design and evaluation". In

10th International Euro-Par Conf. on Parallel Processing, pages 1022–1031, Pisa, Italy, August 31–September 3 2004.

[VMw05] VMware. "ESX server performance and resource management for CPU-intensive workloads". `www.vmware.com/pdf/ESX2_CPU_Performance.pdf`, 2005.

[Zei04] A. Zeichick. "VMware: The virtual desktop, the virtual server". *DevX*, September 28 2004. `www.devx.com/amd/Article/22036`.

TRADEMARKS