Computer Networks 53 (2009) 2873–2887

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

VM^3: Measuring, modeling and managing VM shared resources

Ravi Iyer [*], Ramesh Illikkal, Omesh Tickoo, Li Zhao, Padma Apparao, Don Newell

Intel Corporation, 2111 NE 25th Ave., Mailstop JF2-58, Hillsboro OR 97124, United States

A R T I C L E I N F O

Article history:
Available online 21 July 2009

Keywords:
Virtualization
Consolidation
Shared resources
Virtual platform architectures
QoS

A B S T R A C T

With cloud and utility computing models gaining significant momentum, data centers are increasingly employing virtualization and consolidation as a means to support a large number of disparate applications running simultaneously on a chip-multiprocessor (CMP) server. In such environments, contention for shared platform resources (CPU cores, shared cache space, shared memory bandwidth, etc.) can have a significant effect on each virtual machine's performance. In this paper, we investigate the shared resource contention problem for virtual machines by: (a) measuring the effects of shared platform resources on virtual machine performance, (b) proposing a model for estimating shared resource contention effects, and (c) proposing a transition from a virtual machine (VM) to a virtual platform architecture (VPA) that enables transparent shared resource management through architectural mechanisms for monitoring and enforcement. Our measurement and modeling experiments are based on a consolidation benchmark (vConsolidate) running on a state-of-the-art CMP server. Our virtual platform architecture experiments are based on detailed simulations of consolidation scenarios. Through detailed measurements and simulations, we show that shared resource contention affects virtual machine performance significantly and emphasize that virtual platform architectures is a must for future virtualized datacenters.

1. Introduction

The use of virtualization for consolidation of multiple workloads on to a single platform is growing rapidly in datacenter environments. Virtualization offers the opportunity for better manageability, provisioning and lower cost. Virtual machine monitors or hypervisors from VMware [34], Xen community [36], Microsoft [18], and others manage the virtual machines running on a single platform and ensure that they are functionally isolated from one another. However, from a performance standpoint, it is expected that the performance of each of the virtual machines will be significantly affected by the other virtual machines running on the same platform. Since each of the virtual machines can be running entirely different operating systems and workloads, the overall performance behavior of consolidated scenarios will be significantly different from traditional commercial server workloads that run in a dedicated mode on a platform. As virtualization becomes ubiquitous (with emerging cloud computing and utility computing paradigms), it is imperative that contention for shared resources on each platform between virtual machines is carefully studied. In addition, it is important to be able to project the performance of individual virtual machines in order to allow datacenter administrators to appropriately manage the mapping of virtual machines to available platforms. Finally, it is important to be able to develop architectural techniques to ensure that shared resources can be appropriately allocated to virtual machines running simultaneously based on their importance or their behavior.

The goal of our VM^3 research is to *measure*, *model* and *manage* shared resource contention and its implications on virtual machine performance on a consolidated chip-multiprocessor platform. In this paper, we will cover

[*] Corresponding author. Tel.: +1 503 712 3996; fax: +1 503 264 1544.
E-mail address: ravishankar.iyer@intel.com (R. Iyer).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.04.015

all three aspects of VM^3 by answering the following questions:

- *Measure:* What are the implications of shared resource contention within the platform on virtual machine performance?
- *Model:* How do we decompose the performance impact of shared resource contention when there are multiple resources involved and the overheads of virtualization as well?
- *Manage:* For shared resources (e.g. shared cache, memory bandwidth, etc) that are not exposed to the OS or VMM today, how can the resource management be improved significantly?

Our measurements are based on detailed experimentation with vConsolidate (a server consolidation benchmark [4]) on a consolidated chip-multiprocessor (CMP) server. Our measurement experiments show how a virtual machine performance is affected when it is running with other virtual machines on the same platform. Based on our detailed measurement experiments, we next start to decompose the performance effects on virtual machines into three different categories: (a) the effect of virtualization overheads, (b) the effects of CPU core contention overheads and (c) the effects of cache/memory contention overheads. Through this decomposition, we then formulate a modeling approach to predict the effects of these overheads based on offline profiling experiments. We also show how contention overheads for transparent shared resources (such as cache space and memory bandwidth) are as big a component as virtualization or core contention overheads. Based on this observation, we then propose a novel resource management solution called "virtual platform architectures" or VPAs. Our VPA approach essentially allows the monitoring and allocation of transparent shared resources (cache space and memory bandwidth) to virtual machines (which traditionally only have control over number of cores, memory capacity and IO devices).

In our previous work, we started characterizing virtual machine performance behavior [2] and evaluating the impact of resource interference [16] independently. To our knowledge, a detailed evaluation of consolidation environments that cover measurement, modeling and management of shared resource contention for virtual machines has not been presented earlier. Other than this, this paper is the first to propose a unique approach for modeling as well as management of shared resources for virtual machines.

The rest of this paper is organized as follows. Section 2 presents an overview of virtual machines and describes the shared resource contention problem for consolidation scenarios. Section 3 presents a detailed measurement-based analysis of virtual machine performance. Section 4 presents our decomposition and modeling approach for predicting virtual machine performance. Section 5 presents our shared resource management approach based on a transition from "virtual machines" to "virtual platform architectures". Section 6 covers related work in this area. Section 7 summarizes the paper with our conclusions and a direction for future work in this area.

2. Server consolidation and shared resources

In this section, we will present an overview of virtualization and consolidation. We will also describe the vConsolidate benchmark used in this paper and the CMP platforms considered.

2.1. Server consolidation and vconsolidate overview

Within the last decade, data centers have started employing virtualization solutions [3,12,26,27] to consolidate multiple server applications on the same platform. This was motivated by the need to (a) reduce complexity of managing and interconnecting multiple platforms in the datacenter, (b) improving the overall resource utilization and total cost of ownership by sharing resources and (c) allowing more flexibility to migrate applications and deploy newer ones. Within the next decade, it is expected that a significant percentage of all servers in the marketplace will be running consolidated workloads as opposed to individual applications.

Fig. 1 illustrates the transition from an individual image (single OS) and single server application running on a platform to a virtual machine monitor (VMM) or hypervisor running multiple images (i.e. virtual machines – VMs) on a server platform. This transition becomes especially important as the number of cores integrated into CMP processors [11,13] continues to increase and datacenter-on-chip usage models [15] start to become prevalent to support cloud and utility computing models [1,8,9,19,29,31].

In this paper, we study the implications of server consolidation by employing a recent server consolidation benchmark called vConsolidate [4]. The vConsolidate (vCon) benchmark consists of four key virtual machines (VMs): (a) a compute intensive workload/application, (b) a database workload, (c) a web server workload, and (d) a mail server workload. To emulate a real world environment, an idle virtual machine is added to the mix since datacenters are not fully utilized all the time. The compute intensive VM runs a modified version of SPECjbb2005 [28]. Typically SPECjbb2005 is a cpu intensive workload that consumes as much cpu as it possibly can. However, in this vCon environment, SPECjbb has been modified to consume roughly 75% of the cpu or so, by inserting random sleeps every few milliseconds. The database virtual machine runs Sysbench [30], which is an OLTP workload executing transactions against a MySQL database. The web server VM runs Webbench [35] which uses Apache webserver. The mail server virtual machine runs Microsoft Exchange workload that executes transactions on Outlook with 500 users logged in simultaneously. A vConsolidate configuration as described above with four active VMs and an idle VM comprises a consolidated stack unit (CSU). Fig. 2 illustrates a single CSU configuration. The vCon benchmark also defines various profiles – we have chosen a profile that is a mix of 32-bit and 64-bit VMs including 3 Linux VMs and 2 Windows VMs. The SPECjbb VM (linux 64 bit) has 2 vcpus and 2 GB memory, Sysbench VM (linux 64 bit) has 2 vcpus and 1.5 GB memory, Webbench (32 bit linux) is assigned 2 vcpus and 1.5 GB memory and Mail (32 bit Windows) VM has 1vcpu and 1 GB memory. The idle VM is given 1 vcpu

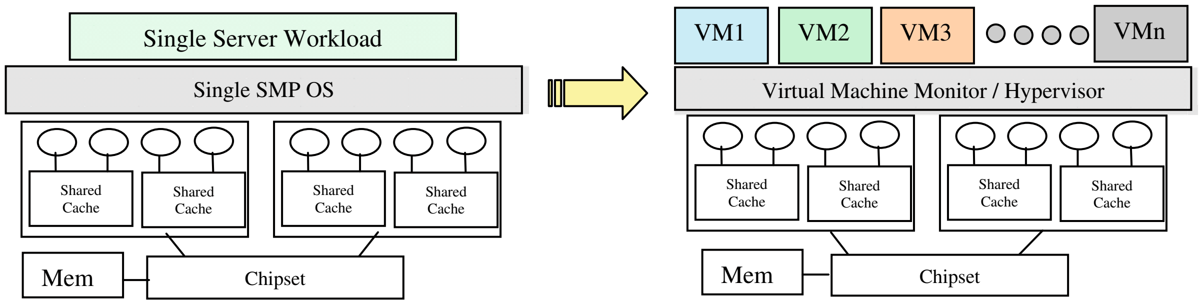


Fig. 1. Towards virtualization and consolidation on CMP Servers.

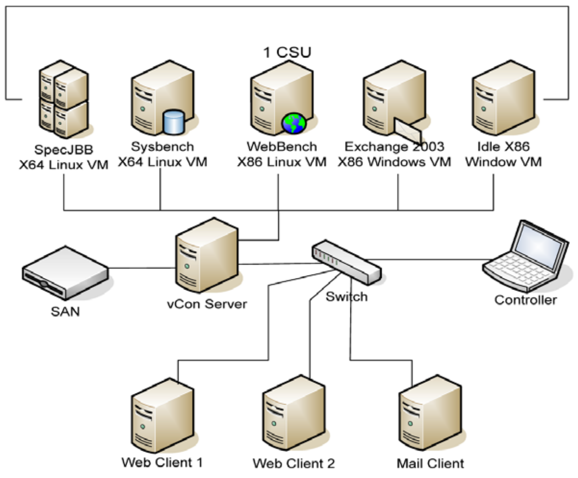


Fig. 2. vConsolidate benchmark components.

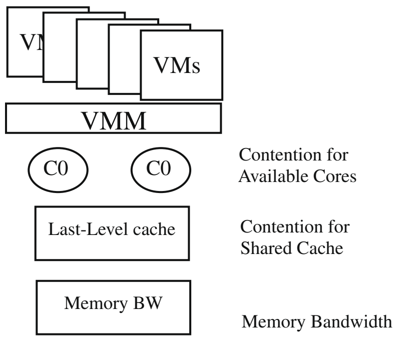


Fig. 3. Shared resource contention and virtual machines.

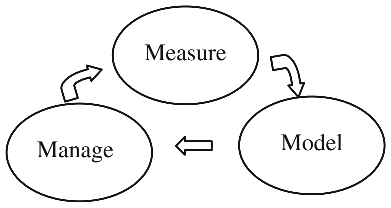


Fig. 4. VM^3 components for shared resource effects.

and 0.4 GB memory. The entire configuration is on a private switched subnet with two clients generating traffic for the Webserver workload and one client generating traffic for the exchange/mail workload. All VMs run on platforms with intel virtualization technology [12,32].

2.2. The VM^3 shared resource problem

As the server consolidation continues to gain momentum, datacenter administrators need to start considering the performance implications of virtualization and server consolidation when making decisions on assigning virtual machines to servers, and capacity planning for resource provisioning. While CMP platforms provide abundant hardware parallelism to achieve server consolidation, running multiple workloads simultaneously introduces contention for shared resources in the platform. A typical CMP processor today consists of multiple cores with one or more shared last-level caches within the processor. Dynamic contention for available cores, shared cache resources and shared memory bandwidth (as shown in Fig. 3) provides better resource usage efficiency, but also introduces performance isolation concerns for the system administrator and data center manager. Without a careful understanding of the effects of resource contention between virtual machines within each platform, it becomes very difficult to satisfy service level agreements that are provided to the customer for their hosted application.

Our VM^3 research was motivated by this key concern of efficient shared resource management within each platform in the datacenter. In this paper, we describe the three key components of our VM^3 research, effectively covering "*measurement*", "*modeling*" and "*management*" of shared resource implications on individual virtual machine performance (see Fig. 4). *Measurement* (offline in this paper) allows us to understand the performance slowdown effects of consolidation on virtual machine performance. *Modeling* (offline in this paper) allows us to decompose the performance slowdown into virtualization effects, CPU core contention effects and shared platform resource (cache, memory in this paper) contention effects. *Management* (online) attempts to improve the current shared resource management approach. Today, VM-based approaches can only control visible shared resources such as cores, memory capacity and I/O devices. In this paper, we show that extending virtual machines into virtual platform architecture (VPA) that encompass the ability to monitor and guide the allocation of traditionally transparent shared resource (cache, memory in this paper) is very important. In the next few sections, we will cover our experimentation on each aspect of VM^3 in significant detail.

3. Measuring resource contention effects

In this section, we first present our measurement methodology and then describe the effects of virtualization and consolidation on virtual machine performance. In particular, we study the shared resource contention effects on virtual machine performance.

3.1. Measurement platform and evaluation tools

Our measurement-based evaluation was conducted on a dual-socket VT-enabled Intel server platform that has two processor sockets (the latest Intel Xeon 5400 series) and is populated with 16 GB of memory. Within each processor socket, there are four cores running at 2.8 GHz. Each pair of cores share a 6 MB cache, adding up to a total of 12 MB on each socket. Fig. 5 illustrates this measurement platform configuration. For our experimentation, to have a saturated platform in consolidated mode, we disabled one pair of cores on each socket and ran our experiments on the remaining four cores. On this platform, we run vConsolidate on top of Xen 3.1. Commonly available tools with Xen such as sar, xentop, xm info, xentrace have been used to get details such as cpu usage from each VM. To get architectural metrics such as CPI (cycles per instruction) and MPI (misses per instruction) we used a tool that reads processor performance counters.

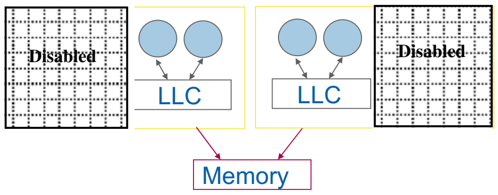


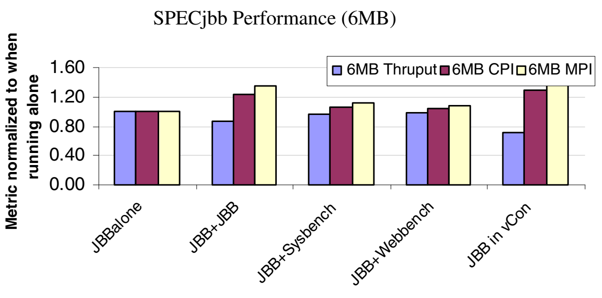
Fig. 5. Measurement architecture/configuration.

3.2. Measurement results and analysis

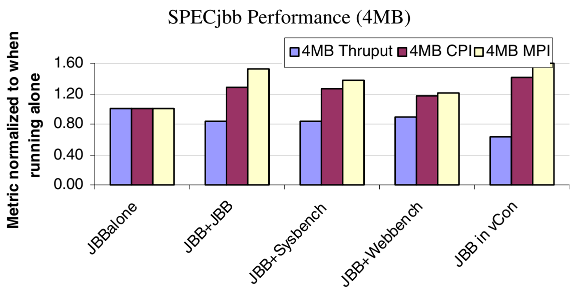
To measure the effects of consolidation, we ran the following configurations: (a) an individual virtual machine (SPECjbb) running alone, (b) the SPECjbb virtual machine running pairwise with each of the other virtual machines that are in vConsolidate and (c) the vConsolidate configuration with all virtual machines running together. Fig. 6 presents the data for SPECjbb2005 and shows how the performance, CPI and MPI change from when it runs alone to when it runs with another workload (pairwise) versus when it runs with all other virtual machines in consolidation. In Fig. 6a, the shared last-level caches are 6 MB in size each. In Fig. 6b, the shared last-level caches are 4 MB in size each. Let's start by analyzing the data in Fig. 6b. When SPECjbb2005 virtual machine runs with another SPECjbb2005 virtual machine, we find that each virtual machine suffers about a 19% loss in performance because it contends for shared cache space and memory bandwidth (but not for cores). When SPECjbb2005 runs with Sysbench or with Webbench, the effect is lower in significance. However, when it runs with all virtual machines within vConsolidate, we find that it loses more than 35% in performance due to both core and cache/memory interference.

The performance loss can be attributed to core, cache [5] and memory interference as follows. The cache and memory interference results in CPI increase, whereas the core contention results in CPU utilization reduction per virtual machine. SPECjbb2005 CPI increases by 40% (implying a performance loss of 30% using the estimation models presented in next section) due to consolidation. To further decompose this performance loss into cache interference and memory interference independently, we measured the memory latency increase to be 10% when going from SPECjbb running alone to SPECjbb running in consolidated mode (resulting in a $\sim 5\%$ performance loss). The remaining gap in performance ($\sim 25\%$) is attributed to cache interference. Fig. 6a shows similar effects on a 6 MB cache. Table 1 shows the additional impact of core interference based on the cpu utilization loss during consolidation. The cpu utilization (in %) is shown in Table 1 and the core interference contributes to about 7–9% of the performance loss (labeled as Delta).

We have also done similar experiments with Sysbench and Webbench to understand the effects of virtual cache



(a) 6MB Last-Level Caches



(b) 4MB Last-level Caches

Fig. 6. Virtual machine performance degradation with consolidation (two different cache size configurations).

Table 1
Core interference impact.

Cpu (%)	Alone	In vCon	Delta (%)
JBB(6 M)	133	121	9
JBB (4 M)	137	127	7
JBB(3 M)	144	130	9
JBB(2 M)	154	140	9

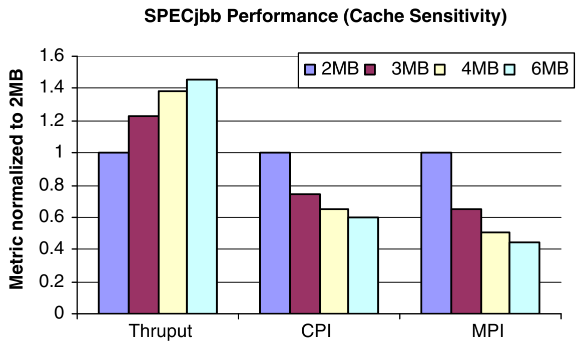


Fig. 7. VM performance sensitivity to cache size.

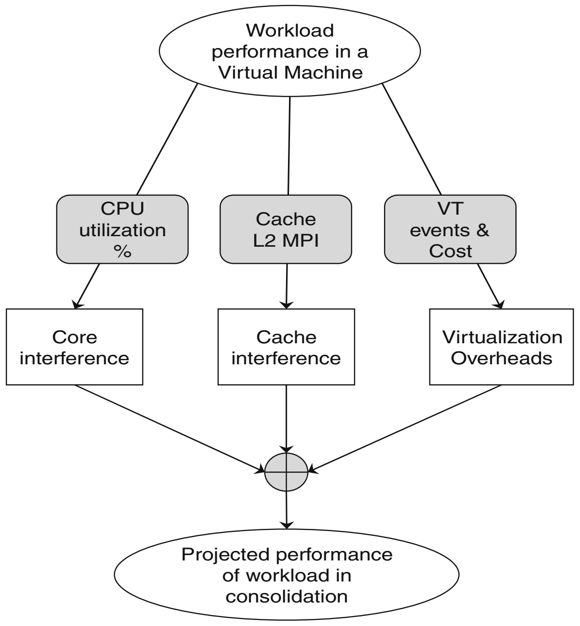


Fig. 8. Model approach for estimating VM performance.

asymmetry (but do not present the data for brevity). Overall, we find that virtual asymmetry due to cache interference has significant impact on each VM's performance.

Since we showed that shared cache interference affects virtual machine performance significantly, we next studied the performance sensitivity of the SPECjbb virtual machine to different cache sizes. Fig. 7 shows the comparison of the overall performance, the CPI and the MPI as a function of the last-level cache (LLC) size (varied as 6 MB, 4 MB, 3 MB, 2 MB). As can be seen in the figure, SPECjbb is quite cache sensitive with the overall performance increasing by as much as 50% as cache size is increased from 2 MB to 6 MB. The performance improvement is reflected in CPI reduction (close to 40%) and the MPI reduction (by as much as 55%). The data also shows (as expected) that the benefits of increasing cache size is higher when going from 2 MB to 3 MB as opposed when increasing from 4 MB to 6 MB.

The above measurement data shows the overheads of core and cache contention on virtual machine performance when consolidated on a CMP platform. The next key challenge is to be able to accurately decompose the resource contention overheads to predict the extent to shared resource contention needs to be managed in consolidated CMP servers. In the next section, we present a modeling approach to decompose the resource contention overheads (core and cache primarily) and show that it is possible to estimate the performance effects with a reasonable degree of accuracy.

4. Modeling resource contention effects

To model the implications of consolidation, we start by proposing a decomposition model that consists of three major components: (a) virtualization overheads, (b) core contention overheads and (c) shared cache contention overheads. Fig. 8 illustrates such a decomposition model that allows us to estimate the potential performance loss when a virtual machine is consolidated with other virtual machines.

Typically, the performance of a workload (like SPECjbb) is measured in transactions per second (the specific metric for SPECjbb is business operations per second or BOPS). This performance is a function of the number of cores, the core utilization, the core frequency, the CPI and the pathlength (instructions per transaction) as follows:

$$\text{Transactions/s} = \frac{\text{Core Util} * \text{Core Freq} * \text{No. of cores}}{\text{CPI} * \text{Pathlength}}$$

The effects of consolidation affect core utilization (due to core contention), CPI (due to platform resource contention such as cache space) and potentially pathlength (if additional virtualization overheads occur). Our decomposition model attempts to capture these overheads with core contention and platform resource contention being the focus for this paper.

4.1. Estimating core contention effects

When running a virtual machine alone on the platform, all the cores in the platform are available to this virtual machine. However, when consolidating this virtual machine with other virtual machines, the contention for CPU cores will affect the available cores. To predict the reduced core utilization, we propose the following simple estimation model:

- Measure each virtual machine's core utilization when running alone (let's call it VM_x-Alone-Util).
- Scale down this core utilization based on the total utilization of all virtual machines running alone and the number of available physical CPUs as follows:

$$VM_x - Const - Util = min\left[VM_x - Alone - Util, \frac{VM_x - Alone - Util * PhysicalCPUs}{VM_{all}\ Alone\ Util}\right].$$

- Estimated performance loss from core contention as follows:

$$Core\ Contention\ Per\,f\ Loss = 1 - \frac{VM_x - Cons - Util}{VM_x - Alone - Util}.$$

For the vConsolidate measurement, we applied the estimation model to platforms configured with different cache sizes (2 MB, 3 MB, 4 MB, 6 MB) as described earlier in Section 3.2. For the estimation model, the utilization was measured with the VMs running alone and per-VM utilization in consolidated case along with the core contention performance loss was calculated using the expressions above. We also instrumented the Xen VMM to measure the actual utilization of the SPECjbb virtual machine and compare it to the estimated utilization. Table 2 shows the utilization (in %) as well as the accuracy (computed as relative error). For example, the estimated utilization is 1.1 cores (110% utilization) for a 6MB configuration, whereas the measured utilization is 1.21 cores (121%) for the same configuration. With such a simplistic model, we find that the error is around 10% or less. It should be noted that such a model can be improved significantly by carefully examining the Xen scheduler and applying weights to virtual machines based on the parameters used for scheduling decisions. We have applied this model to another commercial virtual machine monitor (as well) and find that it comes reasonably close (much like Xen).

4.2. Estimating shared cache contention effects

As described in Section 3.2, the other significant factor for loss in performance for a workload running in consolidation is shared cache interference. Interference in cache is directly related to space contention in cache and its effects on cache miss rate for each of the virtual machines. To estimate cache miss rate increase due to consolidation, we propose the following model (for a pair of cores sharing cache):

- Estimate how often VM_x will run with another VM_y simultaneously on the same shared cache (since there are two cores sharing the cache) in the target platform.

Table 2
Estimating the core utilization.

SPECjbb	6 MB	4 MB	3 MB	2 MB
Est	110	119	123	124
Meas	121	127	130	140
% Error	−9%	−6%	−5%	−11%

This is a straightforward probability exercise based on the utilizations of each of the VMs (in Table 2). Table 3 presents the fraction of SPECjbb VM's execution time that it spends running with another virtual machine contending on a shared cache. In the table Dom0 represented privileged administrative Domain0 in Xen. This domain assists other guest domain in starting and can have direct access to system hardware. For example, a SPECjbb virtual CPU runs with a SysBench virtual CPU sharing the same shared cache for an estimated 30% of its overall execution. On the other hand, it spends an estimated 23% of its overall execution time running with the other SPECjbb virtual cpu.

- Estimate VM_x's effective cache size and MPI when running with VM_y.

We define VM_x's effective cache size as the average cache space occupied by VM_x when sharing cache with another VM_y. To achieve this, we can run pairwise measurements of each of the virtual machines and compare the L2 MPI to that when the virtual machine is running alone. Fig. 9 shows the effect of SPECjbb's MPI normalized to when it was running alone. As shown in Fig. 9, for a 4 MB cache configuration, one SPECjbb virtual CPU's MPI increases by as much as 50% when running with another SPECjbb virtual CPU, whereas it increases by 40% and 20% when running with Sysbench and Webbench virtual CPUs, respectively.

- Estimate VM_x's effective cache size or MPI in consolidated mode.

In order to estimate the overall MPI increase when consolidated with multiple virtual machines (i.e. vConsolidate configuration), we need to take the net effect of individual MPI increases shown above weighted by the execution time fraction also shown before. Based on this approach, Table 4 shows that the estimated MPI (for SPECjbb) comes reasonably close to the measured MPI during consolidation.

The cache contention effect can also be translated to prediction of effective cache size by looking up the cache

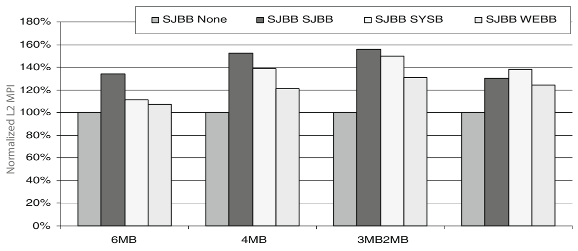


Fig. 9. Estimating MPI increase from pairwise execution.

Table 3
Execution probability of two VMs running cores sharing a last-level cache.

		SJBB (%)	SysB (%)	Webb (%)	Mail (%)	Dom0 (%)
Pairwise execution (%)	SJBB(Meas)	19%	35%	35%	1%	10%
	SJBB (Est)	23%	30%	29%	3%	7%

Table 4
Overall cache contention effect for consolidation.

SJBB	6 M	4 M	3 M	2 M
Meas	0.0053	0.0070	0.0088	0.0111
Est	0.0050	0.0064	0.0087	0.0117

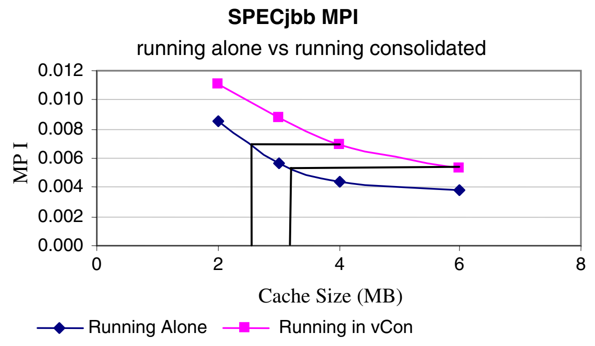


Fig. 10. SPECjbb cache occupancy analysis.

Table 5
Overall performance estimation accuracy.

SPECjbb	6 MB	4 MB	3 MB	2 MB
% Error in performance est.	1%	4%	−4%	1%

size based on offline MPI profiling experiments. For example, Fig. 10 shows the MPI of SPECjbb running alone and in consolidated mode as a function of cache size. By analyzing the miss rate curve (for 6 MB and 4 MB), we find effective cache size (3.2 MB and 2.5 MB), respectively.

4.3. Combining the core and cache contention effects

Above, the performance loss due to core contention and the due to shared cache contention was derived. Taking these two into account, we can predict the performance loss due to consolidation. Table 5 shows the accuracy in overall performance prediction. As shown in the table, the performance prediction accuracy is well within 10% with simple models proposed for core and cache contention alone.

We expect that this simple model proposed in the paper can be enhanced significantly by considering the following:

- Extending the core contention model based on scheduler heuristics implemented in the VMM.
- Extending cache size prediction accuracy by integrating cache size monitoring hardware in the platform.
- Incorporating other shared resource contention overheads such as memory bandwidth contention, multi-threading, etc.
- Although not discussed in this paper, predicting the effects of virtualization based on the type of virtualization technology available (such as [27,33,36]) and the type of virtualization employed by the VMM (para-virtualization, binary translation[33], etc.) is important.

We have now discussed the measurement and modeling potential for virtualization datacenters with an emphasis on shared resource contention implications. In the next section, we will now cover how shared resources such as cache space and memory bandwidth can be better managed to provide better performance isolation and performance differentiation.

5. Managing resource contention with VPA

In this section, we propose virtual platform architectures (VPAs) as an approach to resource management for performance isolation in datacenter servers employing virtualization for consolidation.

5.1. Virtual platform architecture (VPA) overview

A virtual platform architecture is defined as an extension to a virtual machine that includes performance-critical shared resources that are currently invisible to the VMM. Today, a virtual machine consists of number of cores (or virtual cpus), memory capacity and I/O devices (for storage, networking, etc). A VPA includes additional shared resources such as last-level cache space, memory bandwidth, power, etc. In this paper, we experiment with virtual platform architectures that enable monitoring and enforcement of cache space and memory bandwidth (Fig. 11). Monitoring and enforcement of fine-grained resources such as cache space and memory bandwidth requires a unique interaction between the architecture and the operating system. While we limit our discussion to cache and memory bandwidth monitoring and enforcement, the framework proposed is extensible to accommodate other shared resources such as power, TLB, etc., in the platform. In the following subsections, we will describe the architectural support as well as the OS support required to achieve VPA management for performance isolation in datacenter servers.

5.2. Monitoring virtual platform architectures

Major components of VPA monitoring architecture is shown in Fig. 12. In this paper, VPA monitoring tries to achieve two purposes: (a) allows monitoring of transparent resources such as shared cache space on a per VPA basis and (b) reduces the overhead of monitoring by implementing a central monitoring agent in the platform that can automatically log cache space and memory bandwidth per VPA identity. In order to achieve this, the key components of the solution are:

- *VPA identities:* Each platform will support the monitoring of resources on a per VPA-ID basis. This VPA-ID is provided to the platform by the OS/VMM. As a result, it is required that the OS/VMM maintain a VPA-ID per application or virtual machine. Since the number of VPA-IDs are finite (for example: 64), it is also required that the VMM recycles these IDs amongst the active VMs (for

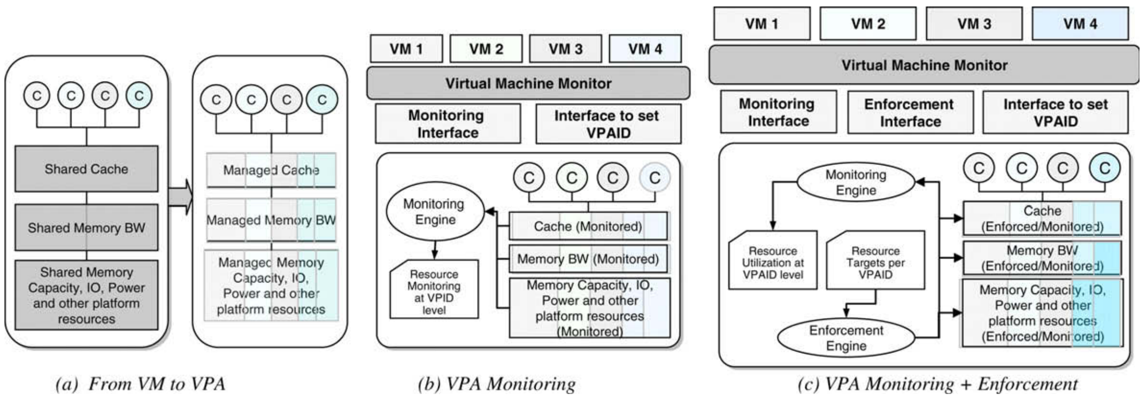

C
Shared Cache
Shared Memory BW
Shared Memory Capacity, IO, Power and other platform resources
Managed Cache
Managed Memory BW
Managed Memory Capacity, IO, Power and other platform resources
(a) From VM to VPA
VM 1
VM 2
VM 3
VM 4
Virtual Machine Monitor
Monitoring Interface
Interface to set VPAID
Monitoring Engine
Resource Monitoring at VPID level
Cache (Monitored)
Memory BW (Monitored)
Memory Capacity, IO, Power and other platform resources (Monitored)
(b) VPA Monitoring
Enforcement Interface
Resource Utilization at VPAID level
Resource Targets per VPAID
Enforcement Engine
Cache (Enforced/Monitored)
Memory BW (Enforced/Monitored)
Memory Capacity, IO, Power and other platform resources (Enforced/Monitored)
(c) VPA Monitoring + Enforcement

Fig. 11. Virtual platform architectures: monitoring and enforcement for transparent shared resources.

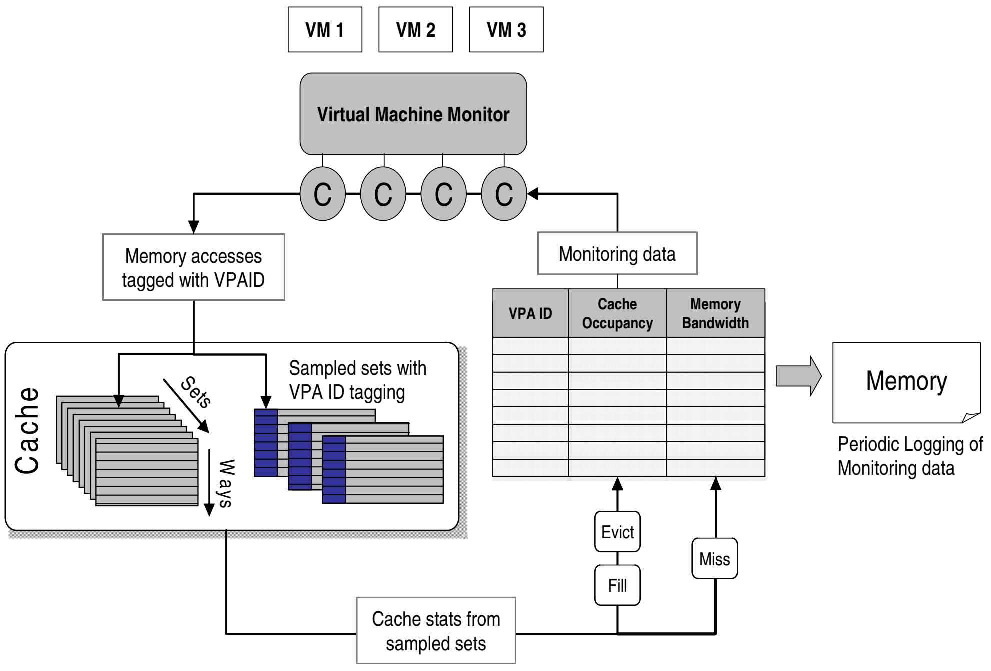

VM 1
VM 2
VM 3
Virtual Machine Monitor
C
Memory accesses tagged with VPAID
Monitoring data
VPA ID
Cache Occupancy
Memory Bandwidth
Memory
Periodic Logging of Monitoring data
Cache
Sets
Ways
Sampled sets with VPA ID tagging
Evict
Fill
Miss
Cache stats from sampled sets

Fig. 12. VPA monitoring components.

example: 128) using recycling/sampling techniques. The sampling technique to achieve this is fairly trivial and will be described in more detail in the evaluation section.

- *VPA register in each core:* In order for the OS/VMM to indicate a VPA-ID to the platform, we introduce a new CPU register in the architectural context for each hardware thread/core. When a VPA is scheduled on the core, the OS/VMM is required to restore the ID of the VPA by storing that value from memory to the VPA register.
- *VPA monitoring agent:* The VPA monitoring agent keeps track of time spent on the core, cache space consumed and memory bandwidth consumed by the VPA-ID in a monitoring table. Every time a new VPA-ID is scheduled on to a core, the VPA monitoring agent receives a message from the core so that it knows when to start and stop accounting for a VPA-ID. Although the VPA monitoring agent can be potentially located anywhere in the die, proximity to shared resources should be kept under consideration.

(i) *Cache occupancy monitoring:* Since cache maintains persistent state, the monitoring of cache state is more difficult than core or memory utilization. For maintaining occupancy of cache per VPA-ID, we need to tag each cache line with the VPA-ID. In order to do so, we ensure that every cache line is associated with the VPA-ID. This can be easily done because the VPA monitoring agent already knows (through core utilization accounting) which VPA-ID is running on which core. Alternatively, it is also possible to include the VPA-ID in every memory request that looks up the cache. One of the key challenges with maintaining a VPA-ID per cache line is the overhead in terms of number of bits and die area. For example, maintaining 64 VPA-IDs requires 6 bits per cache line of 64 bytes which amounts of $\sim 1\%$ overhead in number of additional bits. In order to reduce this overhead further, we employ set sampling [24,37] to reduce the overhead significantly. By tagging less than 10% of the sets in the cache, we can reduce the overhead down to 0.1% but retain a 90+% accuracy which is more that sufficient for monitoring purposes. Through our own detailed experiments, we will actually show that >95% accuracy is possible with less than 5% of the sets tagged in the cache. This makes VPA cache occupancy monitoring more viable. Once a sampled line is tagged in the cache, the cache occupancy count maintained in the monitoring table is essentially incremented when a new line is allocated for VPA-ID and decremented when an existing line is evicted from the cache.

(ii) *Memory bandwidth monitoring:* Memory bandwidth monitoring is relatively straightforward when the monitoring agent is in the last-level cache (LLC). Since memory bandwidth is consumed on all LLC misses and writebacks, the VPA monitoring agent can keep track of bytes consumed during the period of time that a VPA-ID was running. When a VPA-ID is de-scheduled from the core, the total number of bytes are divided by core time and recorded in the table.

- *Access to the monitoring table:* The last step for VPA monitoring is to allow the OS/VMM or other management software access to the data stored in the monitoring table. Traditionally software reads performance monitoring counters in order to access the processor data related to performance events. Following the same approach, it is straightforward to map entries in the table to similar architectural performance monitoring registers so that the OS/VMM can read it at any time. To make it simpler for the OS/VMM to specify which data is of interest, we also support an event description register in each hardware thread. The event description register can be programmed to specify (a) VPA-ID of interest, (b) event of interest (occupancy vs memory bandwidth). Once specified, the core sends a request to read the monitoring table and provides the appropriate data. Although useful for individual queries, a key limitation of such an approach is its intrusive behavior: (a) the monitoring counters have to be programmed and read individually and (b) the monitoring counters may have to be read very often. To address this problem, we also allow the OS/VMM to specify a physical memory address of a log buffer in a register so that the monitoring agent dump the entire table in memory (using a DMA-like operation). It is the responsibility of the OS/VMM software to save or process the log buffer as it gets full. The log buffer also allows out-of-band access for management utilities. Since the log buffer will be read and updated very infrequently (at a time scale larger than one scheduling quantum), the performance overhead of this mechanism is negligible.

5.3. Enforcing virtual platform architectures

Another important part of the VPA architecture is the capability of resource enforcement. For providing performance isolation across VMs, it is important to be able to enforce a specified resource usage per VPA-ID. Today, VMMs can enforce the time spent by a VM on a core and number of cores used by making scheduler changes. However, there is no control provided for allocating shared cache and memory bandwidth appropriately on a per-VM basis. Researchers have attempted to address this issue by incorporating cache and memory partitioning schemes [14,16,17,21–23,21]. In this paper, we adopt a class-of-service based cache and memory allocation mechanism for VPA enforcement with emphasis on supporting a small number of classes of service. The premise behind this choice is that (a) partitioning cache and memory resources across too many VPAs is complex to implement, (b) partitioning cache and memory resources across too many VPAs will negatively affect throughput because of the loss of shared resource use and (c) most importantly, it is expected that the datacenter global management policy can ensure that only a small number of applications that require isolation are mapped onto any given platform.

The key components of enforcing virtual platform architectures for cache/memory subsystems are as follows:

- *Class of service ID:* The OS/VMM is required to associate each VPA with a class of service ID (VPA-CoS-ID) in order to enforce cache and memory allocation. The platform will support a finite number of CoS-IDs that an OS/VMM can discover & choose from.
- *VPA register in each core:* The OS/VMM can indicate the class-of-service for a VPA when scheduling it on the core by storing that value into the VPA register. It should be noted that VPA register for VPA-CoS-ID indication (enforcement) and VPA-ID indication (monitoring) can be the same. The performance overhead of writing one register in the core is negligible since it is done infrequently (only on a context switch).
- *Class of service mapping to cache/memory:* The OS/VMM can indicate the mapping of a class-of-service to the platform through a configuration sequence. This configuration sequence can be static (upon boot time) or dynamic (preferred approach use in this paper). The configuration sequence essentially modifies a set of registers associated with each component (cache CoS config registers and memory CoS config registers in this paper) to indicate the enforcement required per class of service. This will be described below along with the enforcement approach.
- *Class of service enforcement in cache/memory:* Based on the cache and memory CoS registers, the cache and memory subsystem enforces space in the cache and bandwidth in the memory subsystem appropriately. While this is implementation dependent, our approach in this paper is as follows: We adopt a counter-based enforcement approach for cache and bandwidth-reservation for memory as described previously by Iyer et al. in [16]. For cache enforcement, the cache CoS config register contains the cache occupancy threshold that the VPA should not

exceed. In order to enforce this, we modify the replacement policy to take into the account the current occupancy of the class of service in the class. This enforcement is done on a per cache basis (as opposed to on a per set basis) in order to avoid set imbalance issues. For memory bandwidth enforcement, the memory CoS config register contains the bandwidth weight for this particular application. The memory controller maintains *N* different request queues to support *N* classes of service. The memory controller schedules DRAM based on the bandwidth weights and thereby ensures that over a duration of time the ratio of bandwidth provided to the classes of service are equivalent to the ratio of weights.

5.4. Simulation experiments of server consolidation

The measurement experiments described earlier in Section 3 were intended to show the interference effects and performance/resource usage and there by motivate resource management for transparent cache/memory resources. However, since the measurement platform does not have any VPA monitoring and/or enforcement support, we resorted to simulation experiments to evaluate the value and usage of these capabilities.

We employed several multi-threaded server workload traces (8-thread traces of TPC-C, SPECjbb2005, SAP SD/2T and SPECjappserver2004) to drive platform simulations that consist of core models, cache models and memory subsystem models. These workloads were chosen because they represent server benchmarks for back-end database workloads, Java application server workloads and enterprise resource planning workloads. The cache and memory subsystem models supported VPA monitoring and enforcement and the simulator included a central agent model that maintained the monitoring table. Although we employed a trace-driven approach, we ensured that the scheduling of these traces mimicked a typical throughput-based VMM scheduler.

In the platform simulator, we had support for several new capabilities for this work including: (a) support for mapping all threads of the same server workload with a unique VPA-ID, (b) support for mapping all threads of the same server workload with the same class of service IDs, (c) support for a central monitoring agent and monitoring table, (d) ability to log the occupancy and memory bandwidth from the monitoring agent, (e) ability to enforce cache occupancy and memory bandwidth, (f) ability to tag sampled sets in the cache with VPA-IDs and (g) ability to support several heuristics to sample running applications by mapping to finite VPA-IDs.

The results from the following set of simulation experiments will be described in the next section to highlight the value of VPA-based performance isolation and management:

(a) A simple heuristic that efficiently uses a finite set of VPA-IDs and recycles them across a large number of VPAs.
(b) A set sampling approach to monitoring cache occupancy and memory bandwidth with high degree of accuracy.
(c) VPA monitoring in a cluster of servers, each running a customer virtual machine (TPCC) along with a varying number of other VMs.
(d) VPA enforcement using full isolation mode in a cluster of servers, each running a customer virtual machine (TPCC) along with a varying number of other VMs.
(e) VPA enforcement and monitoring using partial isolation.

These experiments are conducted on a server architecture consisting of eight physical cores sharing an 8 M cache and 16 GB of peak memory bandwidth. The number of software threads (within virtual machines) range from 4 to 24 and these are scheduled and de-scheduled on the eight physical cores based on the behavior of a throughput-oriented scheduler. We expect that the results from these experiments effectively highlight the value of VPA monitoring and enforcement effectively for performance isolation in datacenter servers.

5.5. Simulation-based evaluation of VPA monitoring

VPA monitoring depends heavily on the use of a finite number of VPA-IDs and sampling techniques in the VMM to map VPA-IDs to different applications at different times. Our first set of simulation experiments study the efficacy of simple heuristics to achieve this.

5.5.1. Recycling VPA-IDs and sampling VPAs

The heuristics that we experimented with can be explained as follows:

- *Simple time multiplexing (STM):* The most naïve approach is to essentially use a simple time multiplexing techniques. For example, let's assume that there are 25 VPAs and only eight VPA-IDs supported by the platform. This simple time multiplexing approach essentially picks eight VPAs out of 25 at a time and assigns it to eight unique VPA-IDs for a while. The sampling duration was chosen to be several typical time quanta (roughly 100 ms or more for instance). After the first eight VPAs are done with their sampling duration, the eight VPA-IDs are then recycled to the next eight VPAs out of the remaining 17, and so on. The key problem with such an approach is that the VPA-IDs in the cache are not wiped clean before the VPA-IDs are recycled. One potential solution is to support a hardware flush capability. However, this is not practical due to the number of tags that will need to be invalidated and its resulting stall time overheads. It is possible that the STM (without a hardware flush) results in reasonable accuracy if the running applications wipe out the cache within a short time.
- *Gaps-in-time multiplexing* (GTM): In order to address the problem cases (within the previous approach) wherein the applications do not touch much cache at all and as a result stale data from old VPAs are associated with the new VPAs, we introduce a gaps-in-time multiplexing, where we introduce a time gap between the VPA-ID recycling where all running VPAs are assigned a spare

VPA-ID that ends up acting like an approach that wipes the previous VPA-IDs from the cache. For example, for the same 25 VPAs and 8 VPA-IDs, let's add a spare VPA-ID (a 9th one) that the hardware allows the VMM to choose for "wiping". If the first set of eight VPAs are chosen with eight unique VPA-IDs for a sampling duration and then it is followed by a gap duration where all VPAs are assigned the 9th ID, the cache will get filled with almost all cache lines that have the 9th VPA-ID as the ID. Now, when the eight VPA-IDs are recycled after the gap duration to another eight VPAs, the issue of stale data will not occur. This method essentially enables a flush functionality without stalling the execution.

- *LRU or LOU allocation of IDs:* Another approach is to allocate an ID one at a time. In this approach, the key issue is which ID to be recycled when we run out of VPA-IDs. We test two techniques (an LRU policy which picks the victim VPA-ID to be the one that was least recently allocated into the core or an LOU policy which picks the victim VPA-ID to be the one that has the least occupancy in the cache).

Our simulation experiments (see Table 6) show that the simple time multiplexing (STM) technique (25 VPAs with eight VPA-IDs) has an error ranging from 6% to 10% when compared to full accounting (25 VPAs with 25 unique VPA-IDs). This error is not too significant because server workloads that we used are very cache-intensive and quickly wipe the cache anyway within a short time duration. The gaps-in-time approach (GTM) improves the error rate significantly by reducing it to less than 3%. For the server workloads we tested, we found that the LRU/LOU policies also show an error rate close to the GTM techniques (with the LOU marginally better than the LRU). Overall, we note that the gaps-in-time multiplexing technique is most suitable for all cases, especially when there are several workloads that are not cache-intensive and some that are cache-intensive.

5.5.2. Set sampling accuracy

Our next experiment was conducted to test the number of sets we need to sample to have high accuracy in occupancy and miss rate. Fig. 13 shows the data for an 8MB cache with 8192 sets and a set sampling size varied from 8 to 512. There were 32 server threads run on 8 physical cores and each thread was treated as a VPA. The lines in the chart represent the % error at various set sampling sizes for the 32 application threads. The bold line is the average error rate across all these threads.

From the figure, we find that a very small set sampling size (128 sets which is less than 1% of the 8192 sets) is sufficient for greater than 98% accuracy in occupancy monitoring. As we increase the number of set sampling, the error rate reduces. A few threads show a small increase in error rate when going from 32 to 64 sets. This occurs when hot sets for these workloads are chosen for set sampling. Previous work has also shown that a small set sample size is more than sufficient for high accuracy [37] in miss rate comparison. As a result, we were satisfied that a small set sampling structure is sufficient for the rest of our monitoring experiments.

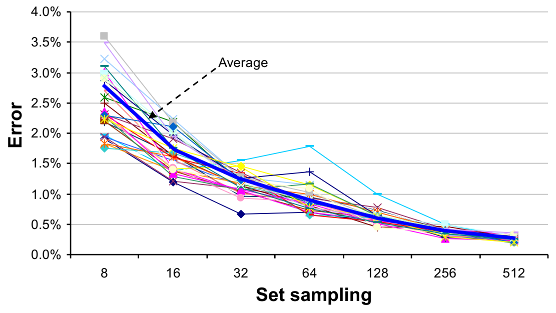


Fig. 13. Accuracy of set sampling techniques.

The next set of experiments was conducted to understand the benefits of VPA monitoring. In this experiment, we ran a varying number of VPAs on eight homogeneous servers. In each of the eight servers, there was one 4-threaded server workload that was held constant, whereas all other server workloads varied in the number of workloads as well as the number of threads within each workload. Table 7 summarizes the configurations on each of the servers.

Fig. 14 shows the resource usage and performance impact of running an instance of a customer VPA on each of the eight servers with different other VPAs running simultaneously. Fig. 14(a) shows the cache occupancy monitored using the VPA monitoring scheme described earlier. As can be observed, the cache occupancy reduces from 100% in the case where the VPA is running alone down to 23% when running on a heavily loaded server. The average cache occupancy across all eight servers is 40%. This value would be used by the OS/VMM (or any other management software) to determine the need for enforcement based on its VPA configuration, since it indicates the average value across all of the servers that the customer VPA was running on. Fig. 14(b) shows the memory bandwidth associated with the customer VPA. The memory bandwidth consumed varies from 0.8 to 1.5 GB/s per thread for this VPA. Running together with other VPAs increases the number of misses (due to contention) as well as reduces the performance. However, since the increase in number of misses is much more than the reduction in performance, the overall memory bandwidth consumption increases. Fig. 14(c) shows the core utilization as a percentage of 8 cores used by the

Table 6
Recycling/sampling accuracy.

	Behavior	Error rate
STM	No. stale data elimination	6–10% error rate (varies significantly)
GTM	Stale data elimination by wiping	2–3% error rate (not much variation)
LRU/LOU	Effect of stale data minimized by picking appropriate candidates	3–4% error rate (not much variation)

Table 7
Workload configuration across eight servers.

Config label	VPAs & Threads
4 T	1 VPA (4 TPCC threads)
8 T	1 VPA (4 TPCC threads) + 1 VPA (4 SJBB threads)
10 T	1 VPA (4 TPCC threads) + 1 VPA (3 SJBB threads) + 1 VPA (3 SJAS threads)
12 T	1 VPA (4 TPCC threads) + 1 VPA (4 SJBB threads) + 1 VPA (4 SJAS threads)
18 T	1 VPA (4 TPCC threads) + 1 VPA (7 SJBB threads) + 1 VPA (7 SJAS threads)
20 T	1 VPA (4 TPCC threads) + 1 VPA (8 SJAS threads) + 1 VPA (8 SAP threads)
22 T	1 VPA (4 TPCC threads) + 1 VPA (4 SJBB threads) + 1 VPA (8 SJAS threads) + 1 VPA (6 SAP threads)
25 T	1 VPA (4 TPCC threads) + 1 VPA (6 SJBB threads) + 1 VPA (8 SJAS threads) + 1 VPA (7 SAP threads)

customer application during the execution. As the server gets more heavily loaded, the core utilization for the customer VPA reduces significantly as expected. Fig. 14(d) shows the impact of cache occupancy reduction and memory bandwidth increase on the IPC (Instructions per Cycle) for the customer VPA. The IPC reduces by as much as 35% due to the reduced resource usage. Overall, Fig. 14(e) shows the performance impact which is a multiplicative effect of the decrease in core utilization and the decrease in IPC.

5.6. Simulation-based evaluation of VPA enforcement

From the monitoring experiments, we observed that the occupancy for the customer VPA was between 25% (2 MB) and 40% (3.2 MB) for all servers where other VPAs were also running simultaneously. Our first enforcement experiment essentially attempts to enforce cache occupancy to a minimum of 3 MB at all times. This is done to support an enforcement model where the customer requires a minimum cache resource of 3 MB ($\sim$ 37.5% of 8 MB). In order to enforce this requirement, we essentially set this VPA to a class of service mapped to 100% threshold (high priority class) and all other VPAs running on the same platform are mapped to a class of service that is either mapped to a 50% threshold (medium priority class) or a 12.5% threshold (low priority class). Since the sum of the medium and low priority class can never exceed 62.5%, the high priority class will get a minimum 37.5% cache space at all times. It should be noted that this 37.5% is reserved even when the VPA belonging to this class is de-scheduled from the core and is waiting to be re-scheduled. As a result, when it is re-scheduled, it enjoys the luxury of a warm cache that is a minimum of 3 MB in size. Fig. 15 shows the benefit of this form of full isolation labeled as QoS_1. As shown in Fig. 15a, the cache occupancy for the customer VPA of interest is always higher than 37.5% in all cases. As a result of this 37.5% minimum occupancy enforcement, the miss rate also improves as shown in Fig. 15b. Overall, the performance in term of IPC shown in Fig. 15c improves by 10–20% as a result of this enforcement.

The second enforcement experiment labeled as QoS_2 in Fig. 15 is an optimization over full isolation called partial isolation. In full isolation, it is required that only one application be allowed to have full access to the cache since any interference from another application of a different class cannot guarantee a pre-specified occupancy at all times. In partial isolation, we allow scenarios where a server runs two VPAs at the high priority class with 100% access threshold. This improves the cache space usage, but could cause problems at times when the two VPAs contend significantly and cause one of them to get occupancy lower than the specification. As a result, we employ this partial isolation approach for a probabilistic enforcement model by monitoring how often the cache occupancy violates the minimum threshold. For example, if the probability of occurrence remains below the probability specified in the contract (say 20% as an example), then the two VPAs

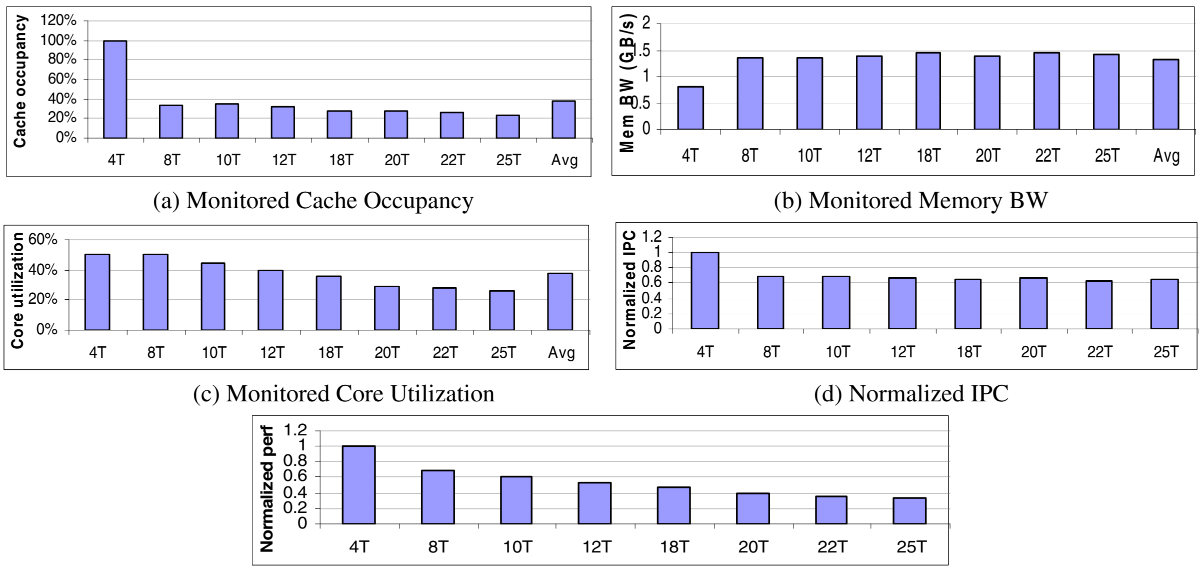
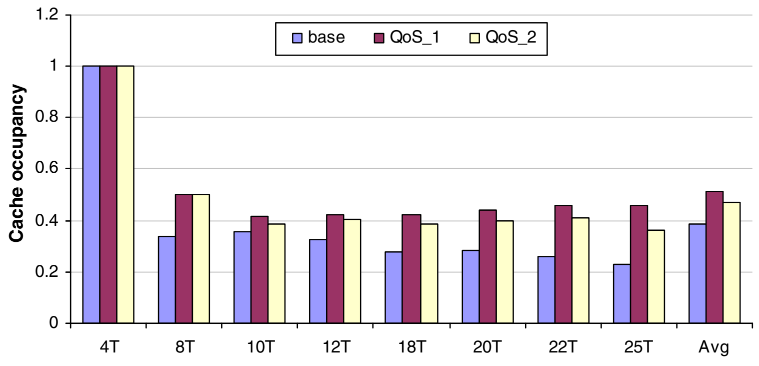
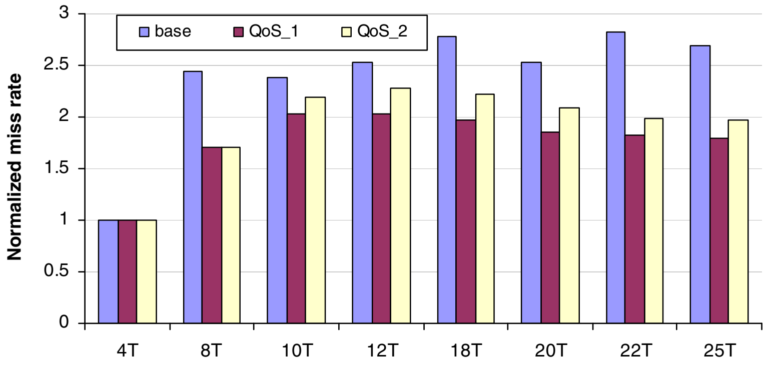


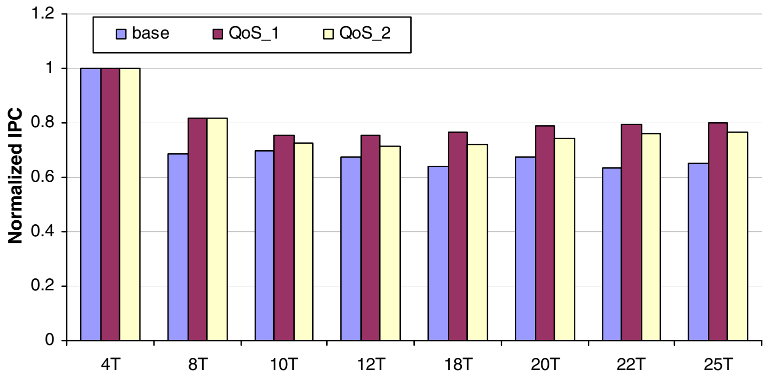
Fig. 14. VPA monitoring: resource usage and performance impact across eight servers for a single application.



(a) VPA Enforcement Impact on Cache Occupancy



(b) VPA Enforcement Impact on Miss Rate



(c) VPA Enforcement Impact on Overall IPC

Fig. 15. Benefits of VPA enforcement.

are allowed to run on the same platform. Otherwise, the global policy manager will be notified so that it can re-distribute these applications on to different servers or full isolation is adopted on the same server.

As an example, Fig. 15 shows the effect of having more than one VPA running in some cases (chosen as 10 T, 18 T, 25 T at random) at high priority along with the customer VPA of interest. From the figure, it can be noticed that although some other VPAs were also run at high priority, the cache occupancy reduces below 37.5% only for one scenario (i.e. 25 T) by just a little bit (36.3% instead of 37.5%). The probability of occurrence in this case is 12.5% (which is below the 20% requirement).

6. Related work

Related work in this area is as follows. Recently there have been a few studies on measurement and characterization of consolidation effects. For example, Cherkasova [6] measure the CPU and IO overheads of virtualization. Jerger et al. [7] have recently studied the degree of sharing in caches and the implications of different scheduling policies on consolidation environments. In this paper, we not only measure the impact of shared resource contention, but we also present a decomposition model to identify the implications of core contention versus cache contention in consolidation environments.

Several researchers have also observed the problem of cache/memory interference [14,16,17,20–23,25] and have proposed resource partitioning techniques to either improve overall throughput, provide fairness or provide QoS. In this paper, we adopt the partitioning techniques to create virtual platform architecture. Our focus, however, in this paper is to understand the use of VPAs for shared resource management and performance isolation in consolidation scenarios. We have shown that this requires both VPA monitoring as well as VPA enforcement.

In terms of VPA, the closest work that has some resemblance to virtual platform architecture is from Nesbit et al. in [22]. It is important to note that VPMs do not consider resource monitoring and a closed loop management with enforcement as a usage model. The VPA-ID approach and CoS-ID approach distinguishes the VPA approach significantly from the VPA approach. In addition we find that employing both monitoring and enforcement techniques is necessary for an efficient performance management solution for the datacenter. Some hardware architectures and software virtualization packages [10,34] provide knobs to fine tune allocation of platform resources at thread, process and VM levels.

7. Conclusions and future work

In this paper we presented our VM^3 research on measuring modeling and managing shared resources and its implications on virtual machine performance. Through detailed measurements using a realistic consolidation benchmark (vConsolidate), we show that contention for shared resource have a significant impact on virtual machine performance and thereby raises performance isolation concerns in virtualized datacenters. We also showed that modeling the impact of shared resource contention can help decompose the overheads in terms of the different shared resources in the platform.

We then showed that a virtual platform architecture abstraction that extends the virtual machine approach with monitoring and enforcement of cache space and memory bandwidth is an absolute necessity for managing shared resources well and providing performance isolation in consolidated virtualized data center environments. We also showed how a VPA solution requires a careful interaction between the OS and the hardware architecture to achieve an efficient solution. The VPA monitoring is achieved using a set of finite VPA-IDs managed by the operating system and provided to the platform. The VPA enforcement is achieved using a smaller set of class-of-service IDs that the OS assigns to a VPA and is enforced by the hardware when indicated. We described the interfaces between the architecture and the OS/VMM to achieve both monitoring and enforcement efficiently. To evaluate VPA, we employed simulation techniques. We showed that the recycling/sampling heuristics to manage VPA-ID to VPA mapping is fairly trivial. We also showed that the use of set sampling can reduce the overhead VPA-ID monitoring while maintaining a high accuracy in terms of reported occupancy. Finally, we performed several monitoring and enforcement experiments to show that the VPA abstraction works well for the consolidation scenarios in today's datacenters.

Future work in this area is as follows. The VM^3 approach of measuring, modeling and managing shared resources can be automated and made more dynamic based on the load conditions in the platform locally as well as globally within the datacenter. In addition, we expect that adding the effect of other resources (power for instance) is quite important. The VPA approach allows us to add other resources quite effectively and we believe that this will open up exciting opportunities for efficient resource management in datacenters in the future.

References

[1] Amazon Elastic Compute Cloud (EC2). <http://www.amazon.com/ec2/>.
[2] P. Apparao, R. Iyer, et al., Characterization and analysis of a server consolidation benchmark, VEE (2008).
[3] P. Barham et al., Xen and the art of virtualization, in: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), October 2003.
[4] J.P. Casazza, M. Greenfield, K. Shi, Redefining server performance characterization for virtualization benchmarking, Intel Technology Journal 10 (3) (2008).
[5] D. Chandra, F. Guo et al., Predicting inter-thread cache contention on a chip multiprocessor architecture, in: Proceedings of the 11th High Performance Computer Architecture (HPCA), February 2005.
[6] L. Cherkasova, R. Gardner, Measuring CPU overhead for I/O processing in the Xen virtual machine monitor, in: Proceedings of the USENIX Annual Technical Conference, April 2005, 30f.
[7] N. Enright Jerger et al., Evaluation of Server Consolidation Workloads for Multi-core Designs, IISWC, 2007.
[8] Google App Engine. <http://appengine.google.com>.
[9] HP Utility Computing. <http://h71028.www7.hp.com/enterprise/cache/308072-0-0-0-121.html>.
[10] IBM Power Processors. <http://www-3.ibm.com/technology/power/>.
[11] Intel Corporation, World's First Quad-core Processors for Desktop and Mainstream Processors. <http://www.intel.com/quad-core/>.
[12] Intel Virtualization Technology Specification for the IA-32 Intel Architecture, April 2005. <http://www.intel.com/technology/platformtechnology/virtualization/>.
[13] Intel Xeon 5400 Series. <ftp://download.intel.com/products/processor/xeon/dc54kprodbrief.pdf>.
[14] R. Iyer, CQoS: a framework for enabling QoS in shared caches of CMP platforms, in: Proceedings of the 18th Annual International Conference on Supercomputing (ICS'04), July 2004.
[15] R. Iyer et al., Datacenter-on-chip architectures, Intel Technology Journal (2007).
[16] R. Iyer, L. Zhao, et al., QoS Policies and Architecture for Cache/Memory in CMP Platforms, the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), June 2007.
[17] S. Kim, D. Chandra, Y. Solihin, Fair cache sharing and partitioning in a chip multiprocessor architecture, in: Proceedings of the 13th International Conference on Parallel Arch. and Complication Techniques (PACT), September 2004.
[18] Microsoft Corporation. <http://www.microsoft.com>.
[19] Microsoft Live Mesh. <http://www.mesh.com>.
[20] O. Mutlu, T. Moscibroda, Stall-time fair memory access scheduling for chip multiprocessors, IEEE/ACM MICRO (2007).
[21] K.J. Nesbit, N. Aggarwal, J. Laudon, J.E. Smith, Fair queuing memory systems, in: Proceedings of the 39th Annual International Symposium on Microarchitecture (MICRO-39), December 2006.
[22] K. Nesbit et al., Providing QoS using virtual private machines, in: First Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI), February 2007.
[23] K.J. Nesbit, J. Laudon, J.E. Smith, Virtual private caches, in: Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA-34), San Diego, California, June 2007.

[24] M.K. Qureshi, Y.N. Patt. Utility-based cache partitioning: a low-overhead, high-performance, runtime mechanism to partition shared caches, in: Proceedings of the Annual International Symposium on Microarchitecture (MICRO), June 2006.
[25] N. Rafique, W.T. Lim, M. Thottethodi, Architectural support for operating system-driven CMP cache management, in: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Technology (PACT 2006), September 2006.
[26] P. Ranganathan, N. Jouppi, Enterprise IT trends and implications on architecture research, in: Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), February 2005.
[27] M. Rosenblum, T. Garfinkel, Virtual machine monitors: current technology and future trends, IEEE Transactions on Computers (2005).
[28] SPECjbb2005. <http://www.spec.org/jbb2005/>.
[29] Sun Network.Com (Sun Grid). <http://www.network.com>.
[30] Sysbench. <http://sysbench.sourceforge.net/>.
[31] Twenty Experts Define Cloud Computing. <http://cloudcomputing.syscon.com/read/612375_p.htm>.
[32] R. Uhlig et al., Intel virtualization technology, IEEE Computer (2005).
[33] <http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf>.
[34] C.A. Waldspurger, Memory resource management in VMware ESX server, in: Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, Boston, Massachusetts, USA, December 9–11, 2002.
[35] Webbench. <http://cs.uccs.edu/~cs526/webbench/webbench.htm.i>.
[36] Xen: The Xen Virtual Machine Monitor. <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/architecture.html>.
[37] L. Zhao et al., CacheScouts: fine-grain monitoring of shared caches in CMP platforms, PACT (2007).

Ravi Iyer is a Principal Engineer with the Systems Technology Lab in Intel's Corporate Technology Group. His current research focus is on SoC and CMP architectures and technologies. Before joining STL, he held positions in the Communications Technology Lab (working on IO acceleration research) and in the Enterprise Products Group (working on server architecture and performance). He received his Ph.D. degree in Computer Science from Texas A&M University. He has filed 25+ patent applications and published 95+ papers in the areas of computer architecture, server design, network protocols/acceleration, workload characterization, and performance evaluation. He has held program committee member positions in various conferences and workshops (MICRO, HPCA, PACT, ICCD, IISWC, etc.). He is an Associate Editor for IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS) and for ACM Transactions of Architecture and Code Optimization (ACM TACO).

Ramesh Illikkal is a Principal Engineer in the Systems Technology Lab at Intel. His research interests are in CMP, server architectures, SoC, virtualization, and memory hierarchies. He received his Masters degree in Electronics from Cochin University of Science and Technology, India. After joining Intel in 1995, he worked on various projects including Platform QoS, IOAT and manycore. He has 20+ patents pending and 25+ technical publications.

Omesh Tickoo received the B.E. degree in electronics and communication from Karnataka Regional Engineering College, Surathkal India and the M.S. and Ph.D. degrees in computer and systems engineering from Rensselaer Polytechnic Institute, Troy NY, USA. He is currently working as a research scientist for Intel Corporation, Hillsboro, OR, USA. His research interests include next generation hardware architectures, network traffic modeling and multimedia streaming over wireless networks.

Li Zhao is currently a research scientist in System Technology Lab, Intel Corporation. She has received her Ph.D. degree in Computer Science from University of California, Riverside in 2005. Her research interests include computer architecture, network computing and performance evaluation.

Padma Apparao is a Senior Researcher in the Systems Technology Lab at Intel. Padma received her Ph.D. degree from the University of Florida and has been working on performance analysis of server workloads. Her current research interest is in SoC architecture and platforms.

Don Newell is a Senior Principal Engineer in the Systems Technology Lab at Intel. His research interests include SoCs, CMPs, server architecture, networking, and I/O acceleration. Newell received a B.S. degree in Computer Science from the University of Oregon.