Shared Resource Monitoring and Throughput Optimization in Cloud-Computing Datacenters

Jaideep Moses (jaideep.moses@intel.com), Ravi Iyer, Ramesh Illikkal, Sadagopan Srinivasan (Intel)
Konstantinos Aisopos (Princeton University)

ABSTRACT
Many datacenters employ server consolidation to maximize the efficiency of platform resource usage. As a result, multiple virtual machines (VMs) simultaneously run on each datacenter platform. Contention for shared resources between these virtual machines has an undesirable and non-deterministic impact on their performance behavior in such platforms. This paper proposes the use of shared resource monitoring to (a) understand the resource usage of each virtual machine on each platform, (b) collect resource usage and performance across different platforms to correlate implications of usage to performance, and (c) migrate VMs that are resource-constrained to improve overall datacenter throughput and improve Quality of Service (QoS). We focus our efforts on monitoring and addressing shared cache contention and propose a new optimization metric that captures the priority of the VM and the overall weighted throughput of the datacenter. We conduct detailed experiments emulating datacenter scenarios including on-line transaction processing workloads (based on TPC-C) middle-tier workloads (based on SPECjbb and SPECjAppServer) and financial workloads (based on PARSEC). We show that monitoring shared resource contention (such as shared cache) is highly beneficial to better manage throughput and QoS in a cloud-computing datacenter environment.

Keywords
Cloud computing, Virtualization, LLC, shared-cache, CMP, Monitoring, data-center, SLA, QoS, Scheduling, Cache Contention, Virtual Platform Architecture

1. INTRODUCTION
The rapidly growing cloud computing model has accelerated adoption of virtualization and consolidation in datacenters. The data center of the past has rapidly evolved into one where there are a large number of heterogeneous applications running within virtual machines on each platform. New cloud Operating Systems like VSphere[18] from VMware have emerged that are designed to holistically manage an entire data center unlike traditional execution environments. The implications of virtualization and consolidation are that multiple virtual machines contend for shared resources on each datacenter platform. Recently, there has been a lot of work published about the issue of contention in the shared cache [1][2][3][4][5] and how this issue can be addressed with various solutions [6][7][8][9][10] that enable monitoring and/or fine-grained resource management such as cache allocation management. While these papers have addressed how each platform could be optimized, our focus is on how we can monitor shared resource usage and contention to improve overall datacenter throughput via migration of virtual machines across platforms. This needs to be done with the Service Level Agreements (SLAs) in mind, because a set of virtual machines (each running on a different platform) collectively represent an application that needs to achieve a level of performance stated in an SLA. Figure 1 shows a typical datacenter illustrating how a platform in the datacenter would be used with various workloads contending for shared resources like the Last Level Cache (LLC) shown here.

In order to improve datacenter throughput and quality of service based on shared resource contention, we investigate the following key aspects in this paper:

(a) **Shared Resource Monitoring:** How can we monitor shared resource usage (taking shared cache as the example) and correlate the shared resource usage back to virtual machine performance?

(b) **VM Migration:** How can we employ shared resource monitoring to migrate virtual machines that are severely affected by shared resource contention?

(c) **QoS and Datacenter Throughput:** What is the overall optimization metric that allows us to determine success or failure of the migration actions taken based on shared resource monitoring?

While there is a lot of previous work in the area of shared resource contention, our contribution is novel as we are the first to propose a simple methodology of using cache occupancy (correlated to instruction throughput) for shared cache environment. Further, we also show how this approach can be used as a critical measurement for determining application/VM migration in a datacenter environment. We also propose a new optimization metric that captures QoS as part of the throughput measure of the datacenter. Through detailed experiments emulating datacenter scenarios (using workloads like TPC-C, SPECjbb, SPECjAppServer, etc), we show that our approach is effective and can provide moderate improvements to QoS and datacenter throughput. We also expect the methodology we put in place is easily extensible for other shared resources (such as memory bandwidth and network). All prior work mainly focused on minimizing contention in the shared cache to improve overall system or datacenter throughput performance. This work is unique as it addresses application/VM scheduling in the context of

SLAs, scenarios where applications/VMs have user specified priorities, and also emphasizes the need for a new metric where QoS is factored into the throughput measure. While mechanisms are already in place to monitor and manage platform resources like I/O and memory, currently there are no mechanisms available for effective management of the shared cache occupancy. We focus our study on shared cache contention which has first-order impact on performance. In our study we assume that there is adequate memory bandwidth enough to avoid second order impact due to change in memory bandwidth consumption. Typically server platforms have at least 20GB/s memory bandwidth and the first order performance impact comes from the LLC contention. We believe that a system wide monitoring mechanism that encompasses memory bandwidth, network and storage is ideal. However we begin with LLC monitoring (missing piece) and intend to extend to resources beyond the LLC for future work.

The rest of the paper is organized as follows. Section 2 provides background and motivation on the problem of shared resource contention by focusing on the Last-Level Cache (LLC). It also shows why shared cache contention can have a significant impact on VM performance and SLA management. Section 3 presents the proposed methodology and the key components of our approach including a new optimization metric that would be needed to evaluate the performance of a datacenter. Section 4 gives a brief description about our simulation infrastructure and presents our analysis and results. In Section 5 we present related work and differentiate our work. Section 6 presents the summary and conclusions drawn in this paper.

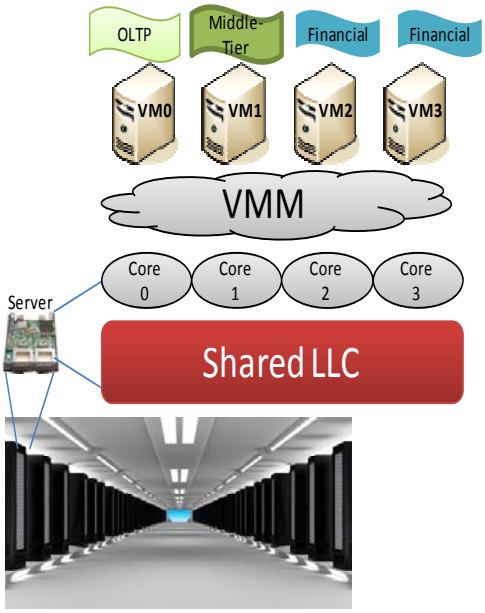


Figure 1: Typical Datacenter Platform and VM Usage

2. BACKGROUND AND MOTIVATION

Based on current trends and recent history in hardware design [13][14], cloud-computing virtualized datacenters of the future will have an increasing number of machines that are based on (Chip Multiprocessing) CMP architectures with multiple cores sharing the same LLC. Large-scale CMP (LCMP) systems will also begin to get employed and initially consolidated with legacy non-CMP systems. It is well understood that LLCs in systems have a first order effect in application performance due to contention in the LLC. We measured the performance of Intel's latest Core2 Duo platform when running all 26applications (in Windows XP) from the SPEC CPU2000 benchmark suite individually and in pair-wise mode. SPEC2000 based results are shown mainly because it has a good spectrum of applications with a variety of cache behavior, and also a lot of researchers are familiar with SPEC2000 characteristics. Figure 2 is intended only to illustrate this background. The x-axis lists each of the 26 applications and the y-axis shows the slowdown in performance when another application (each legend or curve) is also running. For example, the figure shows that mcf runs as much as 5x slower (top curve, leftmost data point) when swim is also running on the platform. This slowdown is entirely due to cache/memory interference between the workloads since there are sufficient cores available and the applications were affinitized to the cores. Overall, we find that 30% of the pairs exhibit 1.2 to 5X slowdown, another 20% of the pairs exhibit 1.1 to 1.2X slowdown and the remaining apps exhibit negligible slowdown.

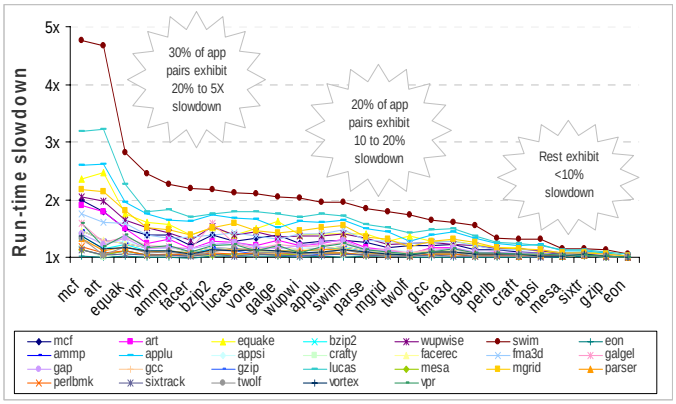


Figure 2. Impact of Cache/Memory Contention

We did similar experiments with other client and server benchmarks which showed similar issues with contention in the shared cache. In this paper we focus primarily on enterprise cloud-computing data centers and hence will use server workloads for motivation and studies. Figure 3 below shows the cache sensitivity of some popular server benchmarks based on simulations we performed in a trace-driven version of CMPSched$im [15].

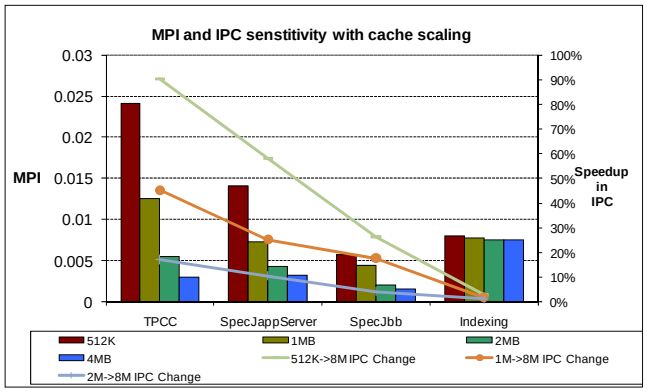


Figure 3. Cache sensitivity of Server Workloads

The X-axis represents the workloads, the primary Y-axis represents the Misses Per Instruction (MPI), and the % speedup in IPC is shown in secondary Y-axis. The first set of bars show that TPCC has a significant reduction in MPI when scaling the cache up from 512K to 8M with MPI values of 0.024 and 0.002 respectively. That translates to a significant 90% speedup in IPC represented by the first point in the top line chart whose value corresponds to the secondary Y-axis.

Similarly SpecJAppServer has a 60% speedup in IPC from 512K to 8M. SPECjbb is also cache sensitive, however the indexing workload does not benefit from cache. To illustrate how the cache sensitivity of these benchmarks would translate to behavior in a cloud-computing virtualized datacenter, we performed a series of simulations with a typical enterprise server machine configuration. Figure 4 shows the performance of the TPCC benchmark while co-running with three other server workloads in a machine with 4 processors sharing the same 4M LLC.

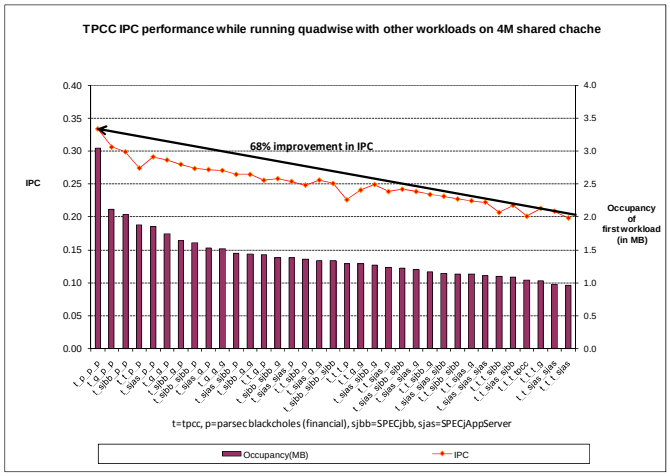


Figure 4. TPCC performance while co-running with other workloads on same shared LLC

The server workloads we use are TPCC, SPECjbb, SPECjAppServer, an indexing workload and parsec blackscholes (parsecblack a financial server workload that is compute intensive). We *assume that all instances of TPCC in this case have identical behavior*. The X-axis enumerates all the possible combinations (without redundancy). The primary Y-axis shows the Instructions Per Cycle (IPC) as a line chart for the first workload (in this case tpcc) in each of the combination and the secondary Y-axis shows the corresponding Occupancy represented as bars. For example, the leftmost bar shows the occupancy of tpcc (3M in this case) when running with three instances of parsecblack and has an IPC of 0.33. The rightmost data-point shows that tpcc while running with two other instances of tpcc and one instance of SPECjAppServer has an occupancy of only 1M due to contention in the shared cache and hence a lower IPC of 0.2. Basically, tpcc would run 68% faster while co-running with three instances of parsecblack as compared to co-running with 2 tpcc's and one SPECjAppServer. This gives us a hint of an opportunity of identifying potential candidates for migration depending upon the priorities that are assigned to various applications/VMs based on SLAs. Clearly, we do not want a high priority application to underperform by running on a machine with one of the combinations to the extreme right of the X-axis in Figure 4. For example, if different instances of TPCC in these mixes have different priorities, one should schedule the high priority TPCC with the application mixes towards the left and the low priority instances with the ones towards the right. We will start addressing several questions in the subsequent sections regarding how do we decide which applications are high priority and which are low, how can a cloud computing OS decide where an application should run depending on it's priority and how do we measure the throughput of the system when there are priorities associated to applications.

3. Proposed MIMe Approach

In this section we will describe our proposed (Monitor Identify and Migrate) MIMe approach which includes the following key components and metrics.

(a) Mechanism used to monitor VM resource usage and identifying VMs that suffer due to resource contention.

(b) Techniques used to identify candidate VMs for migration based on priorities and their behavior to achieve improved weighted throughput and determinism across priorities.

(c) A metric that quantifies the goodness/efficiency of the datacenter as weighted throughput measure.

The following schematic in Figure 5 shows how the key components interact.

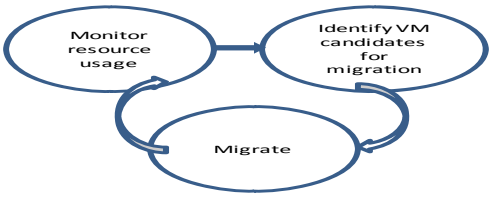


Figure 5. MIMe Key components to improve the efficiency of datacenter weighted throughput

3.1 Monitoring resource usage

The VPA architecture [12] proposed by Ravi et al. can be used to monitor transparent platform resources like cache on a per VM basis. A central monitoring agent that can log resource usage on a per VM basis and expose it to the VMM is a very efficient way of implementing this mechanism.

A VPAID is used to keep track of the cache line's originated from various VMs. VPAIDs are finite and limited, so they are shared and recycled by VMs. Through set sampling techniques [4], the hardware overhead to keep track of VPAIDs is reduced. By tagging a small percentage of sets in the cache, we can reduce the overhead in cache area down to less than 0.5% but still achieve about 90% accuracy that is enough for our purpose. Figure 6 shows a central policy server that acts as a monitoring agent which collects each VM's resource usage and performance is sent to it by each platform in the datacenter. The central monitoring server/agent acts as policy manager and scheduler that determines VMs that are candidates for migration based on resource usage (cache occupancy of each VM) and its corresponding throughput performance (Instructions Per Cycle or IPC).

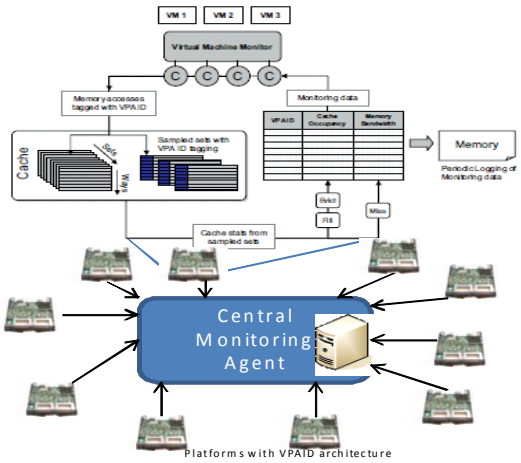


Figure 6. Monitoring using a central policy server

Figure 7 shows a throughput rate curve that can be easily constructed on the fly in a data center. This is derived from the data used in Figure 3.

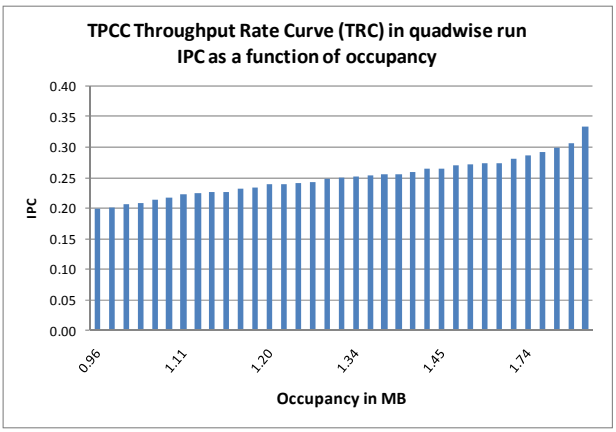


Figure 7. IPC sensitivity for TPCC

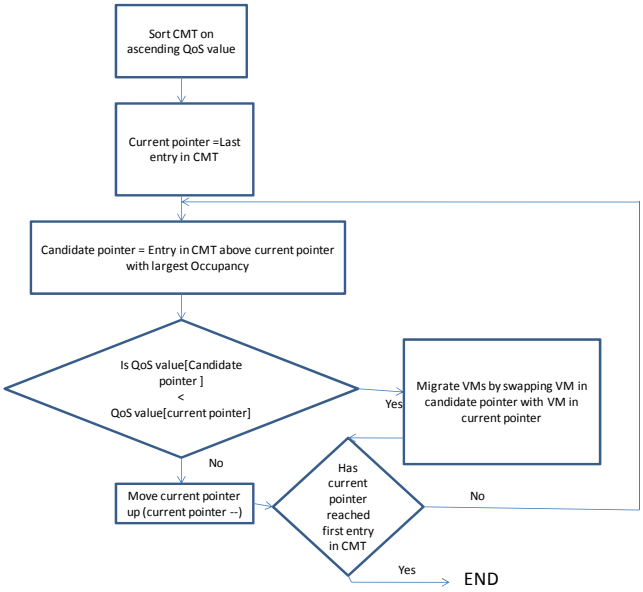
The X-axis shows the occupancies of TPCC in each machine in the datacenter. The corresponding IPCs show a steady increase as the occupancy increases. This clearly shows that occupancy correlates to performance (IPC) for TPCC and hence this information can be used to identify candidates for migration as explained in the next section. We choose IPC as it directly correlates to the execution-time/throughput of an application as long as the application makes progress while executing instructions. The TPCC traces comprises of memory references going into the LLC captured from hardware with optimizations to minimize dependency on I/O. Hence any sensitivity to IPC will directly affect application throughput. Also the simulation based IPC performance was correlated to measurements from hardware.

3.2 Identifying VM candidates for migration

In 3.1 we showed how VPA based architecture can be used to get occupancy and IPC information on a per VM basis to construct a throughput rate curve. This available information becomes very helpful in identifying the potential candidates for migration. As indicated earlier, a VMs priority as agreed upon by an SLA and its behavior (cache sensitivity for example) are two key factors that are used to identify potential VM candidates for migration. Consider a large virtualized cloud computing data center with hundreds of machines that are CMP based, managed by a cloud OS. The machines may come in a variety of cache and processor configurations. Applications/VMs will get scheduled to these machines based on CPU utilization, memory capacity, storage capacity and network bandwidth depending upon SLAs purchased for these applications. Let us consider a configuration wherein machines have identical processors and cache sizes/mappings,. In this instance, we can end up with a scenario wherein an application like TPCC can exhibit a huge variation in performance depending on co-scheduled apps as shown in Figure 7. We assume a simple data structure that maintains a Central Monitoring Table (CMT) with VPAIDs and corresponding occupancies/IPC/other relevant performance

metric and associated application priority. The basic algorithm to identify a candidate VM for migration (as shown in flowchart 1) would be as follows.

1) Sort the CMT by the application priority. This can be a numeric QoS value with the priority of the application being directly proportional to the value (i.e. higher the QoS value, the more important the application is).
2) Assuming the CMT is sorted by QoS value with the largest value (highest priority app) at the bottom of the table, start with the VPAID at the bottom of the table.
3) Find the VPAID in the table with the largest occupancy/IPC above the current entry at the bottom.
4) If the VPAID with the largest occupancy has a lower QoS value than the current VPAID at the bottom of the table, this is a candidate for migration.
5) Migrate the VMs represented by the VPAID at the bottom of the table to the machine with VPAID having the largest occupancy by swapping the two VMs.
6) Update the CMT table to reflect the swap and move to the next entry up in the table. Repeat the steps 3, 4, 5 and 6 until all entries are accounted for and the top of the table is reached.



Flowchart 1: Algorithm to identify candidate VMs for migration

This is basically a simple sort mechanism of specific columns in the CMT making the process a dual sort mechanism (first by QoS value and next based on occupancy relative to QoS value). The mechanism will become clearer with a specific example that will be explained in section 4. The end goal is to ensure that there is no VM of interest that has a higher priority but runs less efficiently than a VM of lower priority after the migrations. This whole process is cyclic as shown in Figure 5. The cyclic ensures that workload phases change or changes in SLAs with customers can be addressed with ease.

3.3 Metric to quantify the efficiency of a datacenter

Now we will address how to quantify the benefits of a migration/scheduling policy at a virtualized cloud-computing data center level. Instructions Per Cycle (IPC) is a commonly used metric that is used to determine the benefits of a new policy or method. Total System IPC or Total Center IPC (For a data center) is also a good measure as it gives a fairly good idea about the overall performance of a system or a data center. However, in a virtualized environment other challenges arise when we want to use a metric to represent overall data center performance. Recent work on server performance characterization [19] for virtualization benchmarking proposed a Vconsolidate concept using weights associated to workload performance. The performance of the virtualized environment can be captured as shown in equation 1 below.

$$\sum_{I=1}^{N} \text{Weight}[i] * \text{WorkloadPerformance}[i]$$

Equation 1: Weighted normalized performance metric

Where, Weight[i] is the weight associated to the i*th* workload or VM. WorkloadPerformance[i] is the normalized performance of the i*th* workload based on the virtualized performance and un-virtualized performance. We modify this concept in the context of QoS and SLAs. We propose a new metric that would incorporate the QoS value as part of the throughput measure. Hence the performance of the virtualized cloud computing environment can be represented by

$$\sum_{I=1}^{N} \text{QoSValue}[i] * \text{IPC}[i]$$

Equation 2: QoS-Weighted throughput performance metric

Where, QoSValue is the QoS value associated to the i*th* workload or VM and IPC is the corresponding IPC of the i*th* workload. Since we use identical machines in our case study, IPC is a sufficient and accurate representation. However, if we want to generalize across platforms, we would have to tailor the above basic metric to capture normalized values of IPC or equivalent metric. In this paper we will stick to the basic new metric we propose in equation 2.

4. RESULTS AND ANALYSIS

We will now briefly explain the methodology and the evaluation framework that was used to obtain the results. We used a simulation based methodology that uses CMPSched$im which is an extension of CMP$im [16], a parallel multi-core performance simulator. CMP$im utilizes the Pin [17] binary instrumentation system to evaluate the performance of single-threaded, multi-threaded, and multi-programmed workloads on a single/multi-core processor. The CMPSched$im simulator originally uses Pin that dynamically feeds instructions and memory references to the simulator. We modified CMPSched$im to be fed by traces so it can be used as a trace-driven simulator.

We used server workload traces for TPCC, Specjbb, SPECjAppServer, indexing workload and parsec that were available in-house. CMPSched$im allows us to get per application cache usage among several other performance metrics like IPC as shown in Figure 4. First we will show how the problem of contention in the shared cache can be addressed in a virtualized data center. In the absence of any type of enforcement mechanisms being available in the hardware to control the cache occupancy, we have to rely only on monitoring information to make scheduling decisions.

In Figure 4, assume that the first workload (in this case tpcc) running in a VM in core0, is the workload of interest for which we want to make sure that contention in the shared cache does not cause it to be slowed down by LP applications. Table 1 shows the occupancy of each instance of tpcc running in VMs and the corresponding IPC with the QoS value. We ignore the priorities of workloads running in Core1, 2 and 3 because migration deals with moving the apps among machines only for Core0. Since the applications in core0 are identical, the migration will not affect the performance of applications in cores 1,2 and 3.

We *assume a simple 4 level priority denoted by values 1, 0.75, 0.5 and 0.25* where 1 is the highest priority followed by 0.5 with lower priority and so on. The values are assigned based on the SLAs with customers, for example, the SLA with the highest quality of service will be assigned 1. Also we *assume a uniform random distribution of these QoS values* across the various instances of tpcc's. In addition, we assume that each application/VMs occupancy can be monitored and is available. Table 1 is sorted by the QoS value. The first column lists 34 instances of tpcc running in core0 of 34 different machines having the same configuration (processor and cache). This type of application to machine mapping can happen if the cloud computing OS takes into account only compute, memory and network usage with no visibility into cache contention.

Core0	Core1	Core2	Core3	IPC	Occupancy in MB	QOS
tpcc0	indexing	indexing	indexing	0.270899	1.52	0.25
tpcc1	indexing	indexing	parsecblack	0.286849	1.74	0.25
tpcc2	indexing	parsecblack	parsecblack	0.306478	2.12	0.25
tpcc3	parsecblack	parsecblack	parsecblack	0.334149	3.05	0.25
tpcc4	sjas	indexing	indexing	0.25619	1.34	0.25
tpcc5	sjas	indexing	parsecblack	0.27225	1.53	0.25
tpcc6	sjas	parsecblack	parsecblack	0.291562	1.86	0.25
tpcc7	sjas	sjas	indexing	0.238714	1.20	0.25
tpcc8	sjas	sjas	parsecblack	0.254059	1.38	0.25
tpcc9	sjas	sjas	sjas	0.222418	1.11	0.5
tpcc10	sjas	sjas	sjbb	0.23154	1.14	0.5
tpcc11	sjas	sjbb	indexing	0.24945	1.27	0.5
tpcc12	sjas	sjbb	parsecblack	0.264918	1.45	0.5
tpcc13	sjas	sjbb	sjbb	0.242182	1.22	0.5
tpcc14	sjbb	indexing	indexing	0.264777	1.44	0.5
tpcc15	sjbb	indexing	parsecblack	0.280173	1.64	0.5
tpcc16	sjbb	parsecblack	parsecblack	0.299236	2.03	0.5
tpcc17	sjbb	sjbb	indexing	0.258371	1.38	0.75
tpcc18	sjbb	sjbb	parsecblack	0.273816	1.61	0.75
tpcc19	sjbb	sjbb	sjbb	0.251129	1.34	0.75
tpcc20	tpcc	indexing	indexing	0.240986	1.29	0.75
tpcc21	tpcc	indexing	parsecblack	0.255999	1.42	0.75
tpcc22	tpcc	parsecblack	parsecblack	0.274576	1.88	0.75
tpcc23	tpcc	sjas	indexing	0.22472	1.13	0.75
tpcc24	tpcc	sjas	parsecblack	0.238735	1.24	0.75
tpcc25	tpcc	sjas	sjas	0.208704	0.97	0.75
tpcc26	tpcc	sjas	sjbb	0.217736	1.08	0.75
tpcc27	tpcc	sjbb	indexing	0.234168	1.17	0.75
tpcc28	tpcc	sjbb	parsecblack	0.248511	1.36	0.75
tpcc29	tpcc	sjbb	sjbb	0.227382	1.13	1
tpcc30	tpcc	tpcc	indexing	0.213103	1.04	1
tpcc31	tpcc	tpcc	parsecblack	0.226079	1.29	1
tpcc32	tpcc	tpcc	sjas	0.198532	0.96	1
tpcc33	tpcc	tpcc	sjbb	0.206792	1.10	1

Table 1. TPCC IPC and Occupancy with QoS values

For example, consider tpcc3 running in core0 (first column), which has the lowest priority QoS value of 0.25. Comparing that to tpcc32 with highest priority QoS value of 1, we see that tpcc3 has occupancy of about 3MB and corresponding IPC of 0.334 compared to tpcc32 with an occupancy of less than 1MB and an IPC of only 0.198. So an HP application may end up running about 70% slower than an LP application. It makes sense to migrate tpcc32's VM to the machine in which tpcc3's VM is running. Note that this migration would not affect the performance of the other applications running in Core1, 2 and 3 because tpcc3 and tpcc32 are identical in behavior. As illustrated earlier in section 3.2, we start with a sorted table (by QoS value) as in Table 1 that can be stored in a data structure in the cloud computing OS. Starting with the machine at the bottom in which tpcc33 is running, we look for the machine in which tpcc running in core0 has the maximum occupancy which happens to be the machine in which tpcc3 is running. We migrate these two applications by swapping them. Next we start with tpcc32 and look for the next largest occupancy which happens to be tpcc2, so we migrate again swapping these and go up the list.

Core0	Core1	Core2	Core3	IPC	Occupanc	QOS
tpcc0	tpcc	tpcc	sjas	0.198532	0.964131	0.25
tpcc1	tpcc	sjas	sjas	0.208704	0.973852	0.25
tpcc2	tpcc	tpcc	indexing	0.213103	1.035851	0.25
tpcc3	tpcc	sjas	sjbb	0.217736	1.082664	0.25
tpcc4	tpcc	tpcc	sjbb	0.206792	1.096372	0.25
tpcc5	sjas	sjas	sjas	0.222418	1.111226	0.25
tpcc6	tpcc	sjas	indexing	0.22472	1.133527	0.25
tpcc7	tpcc	sjbb	sjbb	0.227382	1.134934	0.25
tpcc8	sjas	sjas	sjbb	0.23154	1.140282	0.25
tpcc9	tpcc	sjbb	indexing	0.234168	1.17009	0.5
tpcc10	sjas	sjas	indexing	0.238714	1.197247	0.5
tpcc11	sjas	sjbb	sjbb	0.242182	1.224571	0.5
tpcc12	tpcc	sjas	parsecblack	0.238735	1.240098	0.5
tpcc13	sjas	sjbb	indexing	0.24945	1.267825	0.5
tpcc14	tpcc	indexing	indexing	0.240986	1.285543	0.5
tpcc15	tpcc	tpcc	parsecblack	0.226079	1.294885	0.5
tpcc16	sjbb	sjbb	sjbb	0.251129	1.338929	0.5
tpcc17	sjas	indexing	indexing	0.25619	1.339338	0.75
tpcc18	tpcc	sjbb	parsecblack	0.248511	1.355763	0.75
tpcc19	sjas	sjas	parsecblack	0.254059	1.377979	0.75
tpcc20	sjbb	sjbb	indexing	0.258371	1.379501	0.75
tpcc21	tpcc	indexing	parsecblack	0.255999	1.424704	0.75
tpcc22	sjbb	indexing	indexing	0.264777	1.435105	0.75
tpcc23	sjas	sjbb	parsecblack	0.264918	1.453232	0.75
tpcc24	indexing	indexing	indexing	0.270899	1.519829	0.75
tpcc25	sjas	indexing	parsecblack	0.27225	1.531995	0.75
tpcc26	sjbb	sjbb	parsecblack	0.273816	1.60556	0.75
tpcc27	sjbb	indexing	parsecblack	0.280173	1.64344	0.75
tpcc28	indexing	indexing	parsecblack	0.286849	1.743767	0.75
tpcc29	sjas	parsecblack	parsecblack	0.291562	1.858546	1
tpcc30	tpcc	parsecblack	parsecblack	0.274576	1.879892	1
tpcc31	sjbb	parsecblack	parsecblack	0.299236	2.033106	1
tpcc32	indexing	parsecblack	parsecblack	0.306478	2.116145	1
tpcc33	parsecblac	parsecblack	parsecblack	0.334149	3.051821	1

Table 2. After Migration TPCC IPC and Occupancy with QoS values

Table 2 above shows the final preferred scheduling for all the 34 applications. The goal was to ensure that no higher priority application has occupancy that is smaller than a lower priority application and that is what Table 2 shows. As is the case with most application studies, occupancy for tpcc directly correlates to throughput. In the absence of occupancy monitoring, we can use a similar algorithm by monitoring the IPC and make sure that no higher priority application has an IPC that is lower than a lower priority application. However, with this type of approach we may not know why an application in the lower priority has an IPC higher than an application in the higher priority. The occupancy data helps correlate the slower speed to contention in the shared cache. This basic sorting mechanism is pretty straightforward and easy to implement in a data center. Also, server workloads have a reasonably steady state with no large variations in performance and drastically varying workload phases. So the migration costs would be minimal and thrashing that could occur due to cold-start misses will not be an issue. Hence the overhead of the VM migration is not a concern as the workloads are stable and SLAs don't change dynamically often. Now that we have illustrated the methodology we will examine the results and analyze the performance in detail.

Figure 8 shows the mean IPC for each class (based on QoS value) and it is derived from Tables 1 and 2. This graph captures the benefits of an ideal scheduling wherein higher priority applications have less contention in the shared cache and perform better as they should than lower priority applications.

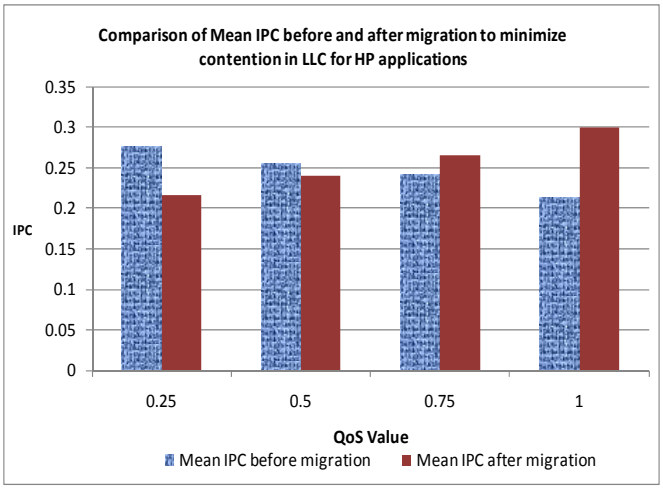


Figure 8: Effect of minimizing contention for HP applications

Figure 8 shows how after minimizing contention in the shared cache for HP applications the mean IPC is more consistent and increases with increasing QoS value. Before migration and when there is higher contention in the shared cache for HP applications, the mean IPC for higher priority applications (QoS =1) is only 0.22 whereas the mean IPC for lower priority applications (QoS = 0.25) is actually higher at 0.276.We use Harmonic Mean as IPC is a rate measure.

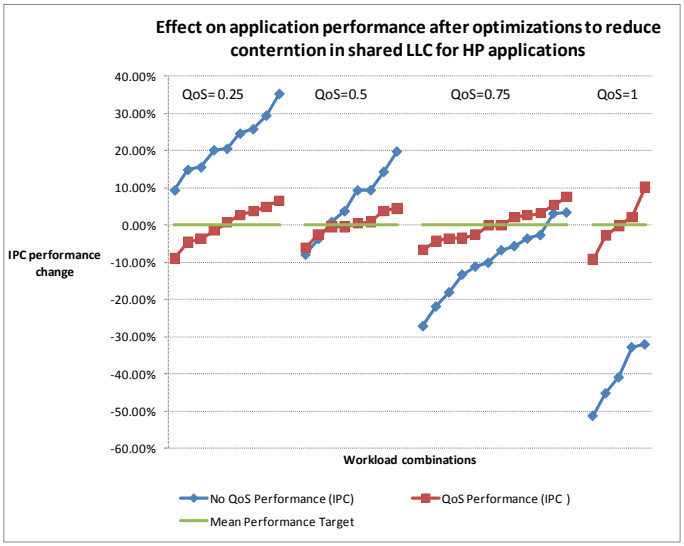


Figure 9. Mean IPC after VM migration for reducing cache contention for HP applications

Figure 9 above shows the application speedup or slowdown before migrations compared to after migrations (for

reducing contention in the shared LLC for HP applications) for TPCC belonging to various classes of service (QoS=0.25, 0.5, 0.75 and 1). The baseline for calculating the speedup or slowdown is the harmonic mean of the IPC for TPCC in each class after migration optimizations. For example, the baseline (Mean IPC after migration) for QoS=0.25 in Figure 9 corresponds to Mean IPC after migration for QoS=0.25 in Figure 8. Examining the S-curves for QoS=1 in Figure 9 we see that almost all HP applications run about 30% to 50% slower before migrations (for reducing contention in shared LLC for HP apps). Also, for QoS=1, it shows that all LP applications run about 10% to 35% faster before optimizations. This is highly undesirable. After migrations for optimizations, we see that applications in all classes show a balanced performance within +-10% of the Mean IPC.

Figure 10 below shows the overall improvement in QoS score for the TPCC workloads/VMs of interest. Before migration based optimizations, the default mapping yields an overall score of 4.9 for the 34 TPCC workloads. However after migration based optimizations, the overall score increases to 5.3 which is about an 8% increase.

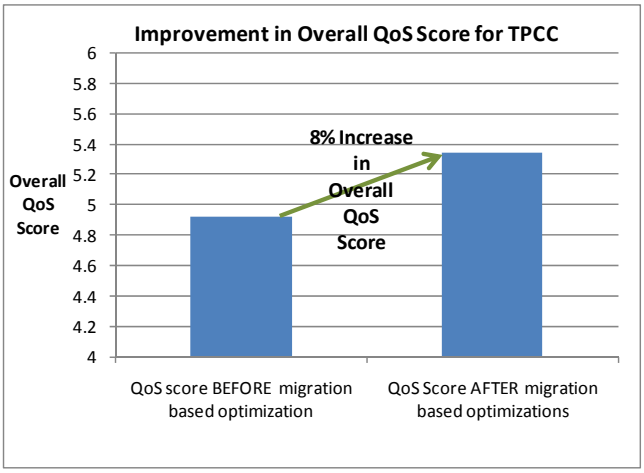


Figure 10. Mean IPC after VM migration for reducing cache contention for HP applications

We repeated the same experiment with SjappServer as the primary workload of interest and observed an increase of about 4.5% in overall QoS Score for SjappServer. It has to be noted that a workload like the indexing workload will not show any noticeable improvement in performance, since it is not cache sensitive as was shown in Figure 3. Before the decision of applying our suggested optimizations is made, it would be helpful to first examine the behavior of the workload by constructing Miss Rate Curves (not shown) or Throughput Rate Curves as explained in section 3.

The basic methodology we suggested in this paper required logically clustering identical machines together and then applying the migration policy. However in real-world datacenters there can be a variety of machines and processors. In such cases, more sophisticated logical clustering mechanisms can be developed. As an example, machines with similar processors but dissimilar cache sizes may also be logically clustered. In fact, with dissimilar cache sizes, there is an opportunity to construct throughput or miss rate curves as the cache sizes are already known. For example there maybe machines that have 2M and 4M caches with possibly identical processors. This would present an opportunity to get the cache sensitivity through miss rate curves. Future systems may also offer enforcement mechanisms that would allow us to control the amount of cache space occupied by each VM. Such mechanisms open an entire spectrum of optimization opportunities. We believe that the basic methodology presented in this paper and the new metric proposed will open a variety of optimization techniques that can be applied to current and future cloud-computing virtualization datacenters.

Finally, in this paper, though we primarily focus on the issue of contention in the shared cache, the methodology can be easily extended to memory bandwidth contention. Once monitoring hooks are available that would allow us to gauge the utilized bandwidth and the effects on memory latency and hence end application performance, optimizations can be done for more efficient scheduling at a datacenter level.

5. RELATED WORK

Related work in this particular area is very limited, though the general problem of addressing contention in the shared cache has been studied extensively. Several papers have observed the problem of contention in shared cache [1][2][3][4][5] and proposed a variety of approaches and solutions [6][7][8][9][10]. However all of these studies and solutions have been primarily focused on manageability on a single machine. Recently, a few studies on characterizing consolidated workloads have been done to measure CPU and I/O overheads in virtualized environments, Cherkasova [21] et al. Enright Jerger [22] have focused on sharing in caches and for better scheduling policies. Nesbit et al. [23] use an enforcement usage model with VPMs that do not consider monitoring. Ravi et al. [12] emphasize a VPA Architecture and propose enforcement solutions. In this paper, we use VPA based architecture and propose a solution in the absence of enforcement mechanisms available in the hardware. We show how identical machines can be logically clustered, and based on VPA monitoring how higher priority applications that we care about are always guaranteed to get more platform resources (cache, in this case study) than lower priority applications. We also propose a new metric that incorporates QoS as the throughput measure.

6. CONCLUSION AND FUTURE WORK

In this paper we showcased exactly why the problem of contention in the shared cache is a critical problem in virtualized cloud computing data centers. Through a case study that uses most commonly studied server benchmarks, we clearly illustrated how high priority applications can suffer if scheduling at a data center level is not done with cache contention in mind. Our work is the first to show a very simple solution based on a VPA architecture that does not rely on enforcement mechanisms to control cache usage. We also emphasize why this problem should not be ignored and how it can be solved without waiting for enforcement mechanisms to be available in the shared LLC. Future work in this area would involve incorporating memory bandwidth as part of the VPA architecture. With VPA monitoring capability, other interesting scheduling optimizations are also possible. Future work would involve detailed profiling of VMs to guide scheduling decisions. Once enforcement capability is available, highly sophisticated techniques that combine the benefits of monitoring and enforcement for cache, memory bandwidth and also power can be very efficiently used in future cloud-computing data centers.

7. ACKNOWLEDGEMENTS

We sincerely thank Aamer Jaleel (Intel) for his valuable contributions to the simulation infrastructure.

8. REFERENCE

[1] L. Hsu, S. Reinhardt, et al., "Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource", PACT, Sept 2006.

[2] H. Kannan, F. Guo, L. Zhao, et al., "From Chaos to QoS: Case Studies in CMP Resource Management," dasCMP/Micro, Dec 2006.

[3] C. Liu, M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," 10th IEEE Symposium on High-Performance Computer Architecture, Feb. 2004.

[4] M. K. Qureshi and Yale N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches", MICRO 2006

[5] S.S.Pinter, M.Zalmanovici, "Data Sharing Conscious Scheduling for Multi-threaded Applications on SMP Machines," Euro-Par 2006

[6] R. Iyer, "CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms," ICS'04.

[7] R. Iyer, L. Zhao, F. Guo, R. Illikkal, D. Newell, Y. Solihin, L. Hsu and S. Reinhardt, "QoS Policies and Architecture for Cache/Memory in CMP Platforms", the ACM SIGMETRICS

[8] P. Petoumenos, G. Keramidas, S. Kaxiras, and E. Hagersten. Statshare: A statistical model for managing cache sharing via decay. In 2nd Annual Workshop on Modeling, Benchmarking and Simulation (2006).

[9] N. Rafique, W.-T. Lim, and M. Thottethodi. Architectural support for operating system-driven cmp cache management. In PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques, pages 2–12, New York, NY, USA, 2006.

[10] P. Petoumenos, G. Keramidas, H. Zeffer, S. Kaxiras, and E. Hagersten. Modeling cache sharing on chip multiprocessor architectures. 2006 IEEE International Symposium on Workload Characterization

[11] David K. Tam Reza Azimi Livio B. Soares Michael Stummm. RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations. Fourteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)

[12] Ravi Iyer, Ramesh Illikkal, Omesh Tickoo, Li Zhao, Padma Apparo, and Don Newell: VM3: Measuring, modeling and managing VM shared resources. Computer Networks, Volume 53, Issue 17, Virtualized Data Centers, December 2009

[13] Intel Xeon 5500 Series, http://www.intel.com/Assets/en_US/PDF/prodbrief/322355.pdf

[14] AMD Phenom II, http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-key-architectural-features.aspx

[15] J. Moses et al: CMPSched$im: Evaluating OS CMP interaction on shared cache management, ISPASS 2009

[16] A. Jaleel et al: CMP$im- A Pin-Based On-The-Fly Multi-Core Cache Simulator, Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS) co-located with ISCA 2008

[17] Pin-A dynamic binary instrumentation tool http://rogue.colorado.edu/pin

[18] VSphere: VMWare VSphere Cloud OS. http://www.vmware.com/products/cloud-os/

[19] Jeffrey P. Cassaza, Michael Greenfield, Kan Shi. Redefining server performance characterization for virtualization benchmarking. Intel Technology Journal, Intel Virtualization Technology, Volume 10, Issue 3, August 2006

[20] Vikram Makhija, Bruce Herndon et al, VMmark: A scalable benchmark for virtualized systems. VMWare technical report, Sept 2006

[21] L. Cherkasova, R. Gardner. Measuring CPU overhead for IO processing in the Xen VMM: Proceedings of the USENIX Annual Technical Conference, April 2005

[22] N. Enright Jerger et al. Evaluation of Server Consolidation Workloads for Multi-Core Designs, IISWC 2007

[23] K. Nesbit et al., Providing QoS using virtual private machines. Workshop on CMP-MSI, 2007