Resource Contention Detection and Management for Consolidated Workloads

J. Mukherjee
Dept. of ECE,
Univ. of Calgary, Canada
Email: jmukherj@ucalgary.ca

D. Krishnamurthy
Dept. of ECE,
Univ. of Calgary, Canada
Email: dkrishna@ucalgary.ca

J. Rolia
HP Labs
Palo Alto, CA, USA
Email: jerry.rolia@hp.com

C. Hyser
HP Labs
Palo Alto, CA, USA
Email: chris_hyser@hp.com

***Abstract*—Public and private cloud computing environments typically employ virtualization methods to consolidate application workloads onto shared servers. Modern servers typically have one or more sockets each with one or more computing cores, a multi-level caching hierarchy, a memory subsystem, and an interconnect to the memory of other sockets. While resource management methods may manage application performance by controlling the sharing of processing time and input-output rates, there is generally no management of contention for virtualization kernel resources or for the memory hierarchy and subsystems. Yet such contention can have a significant impact on application performance. Hardware platform specific counters have been proposed for detecting such contention. We show that such counters are not always sufficient for detecting contention. We propose a software probe based approach for detecting contention for shared platform resources and demonstrate its effectiveness. We show that the probe imposes a low overhead and is remarkably effective at detecting performance degradations due to inter-VM interference over a wide variety of workload scenarios. Our approach supports the management of workload placement on shared servers and pools of shared servers.**

I. Introduction

Large scale virtualized resource pools such as private and public clouds are being used to host many kinds of applications. In general, each application is deployed to a virtual machine (VM) which is then assigned to a server in the pool. Performance management for applications can be a challenge in such environments. Each VM may be assigned a certain share of the processing and input/output capacity of the server. However, servers still have other resources that are shared by VMs but that are not managed directly. These include virtual machine monitor (VMM) resources and the server's memory hierarchy. VMs may interfere with each others' performance as they use these resources. Performance degradation from contention for such resources can be significant and especially problematic for interactive applications which may have stringent end-user response time requirements. We present a method for detecting contention for such resources that is appropriate for both interactive and batch style applications. The method supports runtime management by continually reporting on contention so that actions can be taken to overcome problems that arise.

Detecting contention for shared but unmanaged resources is a key challenge. In cloud environments, management systems typically do not have access to application metrics such as response times. Even if they did it would be difficult, over short time scales, to infer whether variations in an application's response times are normal fluctuations due to the nature of the application or whether they are due to interference on the server. Others have reported success at detecting such contention by using physical host level metrics, e.g., CPU utilization, Clock Cycles per Instruction (CPI), and cache hit rates, to predict performance violations of applications running within VMs [5], [6], [3], [14]. However, these approaches focus on scientific and batch applications. Such applications do not have the high degree of request concurrency or OS-intensive activity that is typical for interactive applications, e.g., high volume Internet services. We show that the existing approaches are not always effective for recognizing contention among VMs in environments that host interactive applications.

We propose and evaluate an alternative approach for detecting contention. This approach makes use of a probe VM running a specially designed low overhead application. The probe executes sections of code capable of recognizing contention for various unmanaged resources shared among the VMs. Baseline execution times are collected for these sections of code while executing the probe VM in isolation on a given server. These values can be continuously compared with those gathered when the probe runs alongside other VMs on the server. Any statistically significant deviation between these sets of measures can be reported to a management controller with information on the type of resource contention that is present, thus allowing a management controller to initiate actions to remedy the problem such as migrating a VM.

In this paper, we implement a probe designed to identify problems related to high TCP/IP connection arrival rates and memory contention. A method is given for automatically customizing the probe for a host by taking into account the host's hardware and software characteristics. Using applications drawn from the well-known RUBiS [6] and DACAPO [7] benchmark suites, we show that the probe is remarkably effective at detecting performance degradations due to inter-VM interference. Furthermore, the probe imposes very low overheads. In the worst case scenario, it caused 1.5% additional per-core CPU utilization and a 7% degradation in application response time performance.

The remainder of the paper is organized as follows. Section II gives a brief introduction to modern server architectures

978-3-901882-50-0

and VMMs. Section III describes the design of the probe. Section IV demonstrates the effectiveness of the approach. Related work is discussed in Section V. Section VI offers summary and concluding remarks.

II. Server Architectures and Virtual Machine Monitors

Modern processors from Intel and AMD have adopted socket based architectures with non-uniform memory access (NUMA) between sockets. Each socket itself is a physical package with a connection to its own local, i.e. "near", memory subsystem and an inter-socket communication connection possibly to other sockets and "far" memory, i.e., remote memory, attached to other sockets in the system. It contains multiple processing cores all sharing a large L3 cache to limit transactions leaving the socket. The cores in Intel processors may additionally implement hyper-threading by creating two logical cores (separate architectural state, but shared execution units) introducing additional shared resources subject to contention.

Modern operating systems such as Linux and Windows have process scheduling and memory allocation support for NUMA hardware. Virtual machine monitors (VMM) such as Kernel Virtual Memory (KVM), being based on the Linux operating system, provide a similar level of support as well. The key characteristic of such mechanisms is the ability to pin processes and VMs to specific cores. Furthermore, a process or a VM can be configured using these mechanisms to use only the local memory. This minimizes contention on the inter-socket communication connections due to L3 cache miss traffic. From a management perspective, this motivates the need to pin a VM to a single socket and assign resources to it from within that socket. We note that VMM kernel resources and caches represent unmanaged resources since there are typically no mechanisms to share these in a controlled manner between VMs.

III. Probe Design

The probe we introduce has two parts. The first part is the sensor application that runs in a VM and makes use of shared server resources. We denote this as the probe sensor. We require one VM probe sensor per socket. The second part is a probe load generator. It causes the probe sensor to execute at specific times and measures its response times. The load generator is deployed on a separate host. The load generator is implemented using an instance of the httperf tool [11]. Measurements on our test setup reveal that httperf can support 10000 connections per second (cps) per core. Based on our final probe design outlined in IV-D3, this suggests that each core on a load generator host can support the monitoring of up to 400 probe sensors and hence managed sockets.

For this study, we narrow our focus to problems arising from high TCP/IP connection arrival rates and inter-VM memory contention. Accordingly, the probe sensor alternates between two phases of processing. The first phase focuses on sensing contention at the VMM related to establishing TCP/IP connections. The second phase deals with contention for the memory hierarchy. We refer to these as the connection and memory phases respectively. The probe load generator controls which phase the probe sensor is in and is able to communicate any inferred performance degradations to management controllers.

In the connection phase, an Apache Web Server instance within the probe sensor VM is subjected to a burst of c Web cps over a period of t_1 seconds. The Apache instance services a connection request by executing a CPU-bound PHP script. In the memory phase, the probe sensor VM executes a script that traverses an array of integers of size n for t_2 seconds, where n is related to the L3 cache size for the socket. Specifically, the value of n is chosen so as to cause a L3 miss rate of 0.1. The values of c, t_1, n, and t_2 need to be carefully tuned for each type of server in a resource pool such that the probe imposes low overhead while having the ability to identify contention among VMs.

To tune the probe for a particular server, we employ static information about host socket cache size to choose n and a set of micro-benchmark applications. Each micro-benchmark is known to stress a particular shared host resource. To tune one of the probe's phases, the number of its corresponding micro-benchmark VMs is increased until performance degradation is observed. This process is repeated again but with the probe in place. Parameters for the probe, i.e., c, t_1, and t_2 are optimized in a one factor at a time manner until the probe has low resource utilization yet correctly reports the degradations as incurred by the micro-benchmark VMs. Details of the tuning process are given in Section IV-D3.

IV. Case Study

Our experimental design has four parts. The first part demonstrates the importance of socket affinity for workload management. The second part focuses on OS and hardware counters. It assesses their effectiveness at detecting contention for shared but unmanaged resources. The third part demonstrates the effectiveness of the probe approach at detecting such contention for the micro-benchmark applications and the tuning of the probe using the micro-benchmark applications. Finally, the fourth part demonstrates the effectiveness of the tuned probe when used with well known benchmarks of interactive and batch workloads that are more representative of those applications likely to be executed in cloud environments. The measurement setup we employ is described in Section IV-A. Sections IV-B through IV-E present the results of the four parts of the study, respectively.

A. Measurement Setup and Micro-benchmarks

For all our experiments, we use a 2 socket, 4 cores per socket AMD Shanghai Opteron 2378 Processor server. The server has private per-core L1 (256 KB) and L2 (2 MB) caches and a shared per-socket L3 cache (6 MB). It has an 8 GB NUMA memory node attached to each socket. The server has an Intel 82576 Gigabit NIC with two 1 Gbps ports. The two sockets are connected through a HyperTransport 3.0 link.

We present experiments where the server is used with and without virtualization. For virtualization we used KVM version qemu-kvm-0.12.5. We configured each VM with 1024 MB of memory, 1 VCPU, 12 GB disk space, and a "Bridge" type network interface. The physical host as well as the VMs executed the Linux Ubuntu 10.10 (kernel version 2.6.35-22-generic) OS. The Linux-Apache-MySQL-PHP (LAMP) stack is used with Apache version 2.0.64 and MySQL version 5.1.49-1 in both the non-virtualized and virtualized experiments.

We tuned Apache, MySQL, the guest OSs and the VMM to support tests with a large number of concurrent TCP/IP connections. In particular, the connection pool sizes of Apache and MySQL were configured to handle up to 1024 concurrent connections. We also used Linux IRQ affinity mechanisms to ensure that network interrupts processing is balanced across the cores in the system.

The server is connected to 6 load generator PCs through a 48 port non-blocking Fast Ethernet switch that provides dedicated 1 Gbps connectivity between any two machines in the setup. The load generator PCs are non-virtualized. Each load generator is responsible for generating synthetic requests to one application VM involved in an experiment. Our experiments involve both interactive and memory-intensive applications. For an interactive application, an instance of the httperf tool is executed on a load generator to submit multiple concurrent requests to the application and record request response times. For a memory-intensive application, a custom script is executed to collect application execution times. We also collect several OS-level and hardware-level measurements for the server under study. The Linux collectl tool was used to measure CPU and disk utilization of the Web server at a 1-second granularity. To get deeper insights into the results, we used the Oprofile tool to monitor hardware events such as the misses at various levels of cache hierarchy.

We consider two Web server micro-benchmarks and one memory micro-benchmark. The Web server micro-benchmarks involve an Apache Web server that serves statistically-identical requests. In one case the request processing is very light weight – we refer to this as the kernel-intensive workload. It mimics a workload that causes significant TCP/IP kernel activity by serving a single static 4KB file. In another case the request processing causes additional user space processor usage – we refer to this as the application-intensive workload. Each request invokes a PHP script on the web server that causes an exponentially distributed CPU service demand of 0.02 ms. It has approximately ten times the user space level processing per connection as the kernel intensive workload. The memory micro-benchmark we use is the RAMspeed micro-benchmark of the open-source Phoronix benchmark suite [8], which is designed to cause significant cache misses.

B. Management by Socket

The first part of our study considers the relationship between workload performance and workload placement within a multi-socket server. For these experiments we do not employ a VMM. We consider three scenarios: host affinity, socket affinity, and core affinity. For the scenarios, the Linux CPU Hotplug utility is used to enable and disable cores on sockets. Host affinity uses the default Linux scheduler to manage process scheduling across cores enabled on two sockets. Socket affinity restricts a workload to a specific socket but allows it to freely migrate among the cores in that socket. Core affinity binds each workload to its own distinct core using the Linux taskset utility.

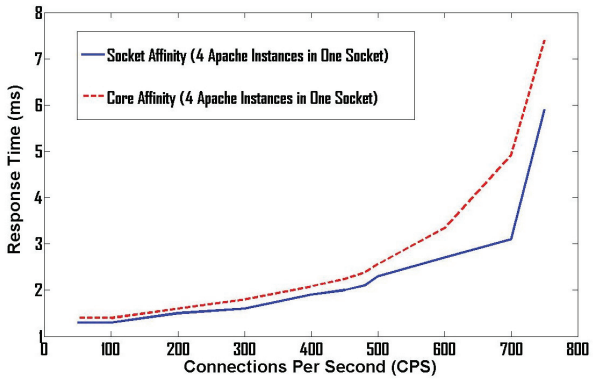

Fig. 1. Socket Vs Core Affinity for Application-Intensive Workloads

A first experiment showed that core affinity outperformed host affinity for both kernel and application intensive workloads by enabling up to 40% more application level throughput. These are expected results given that host affinity can incur near-far memory access latencies. A second experiment compares core affinity with socket affinity for four statistically-identical instances of the application intensive workload. One socket is used and all four cores are enabled. Figure 1 shows that socket affinity outperforms core affinity. From other results we are unable to show due to space limitations, the difference is even more extreme for the kernel intensive workload than the application intensive workload. These results suggests that workloads with resource requirements that can be satisfied by a single socket will perform best with socket affinity.

C. Effectiveness of Counters at Detecting Contention

This second part of the study assesses the effectiveness of commonly used hardware and OS counters [5] at detecting contention among interactive applications sharing a physical server. We consider cycles per instruction (CPI), L1, L2, and L3 cache miss rates, and CPU utilization values. Sections IV-C1 and IV-C2 consider scenarios without and with the use of virtualization.

1) Without Virtualization: This section considers several different assignments of Apache instances to cores on the server. We show that the different assignments can result in different performance behavior and that commonly used hardware and OS counters do not provide significant insight into the differing behavior. We host up to 6 Apache instances on the server. The Apache instances support statistically identical workloads. Each configuration is denoted by (x,y) where x and

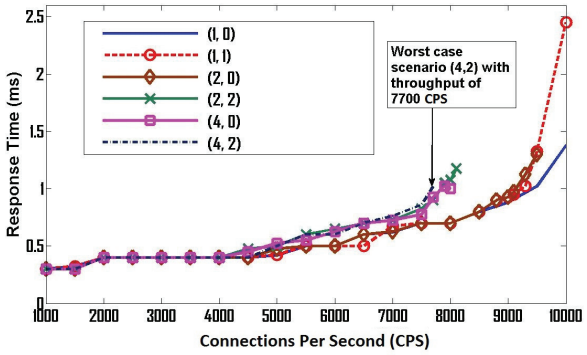

Fig. 2. Kernel-Intensive Workload Performance

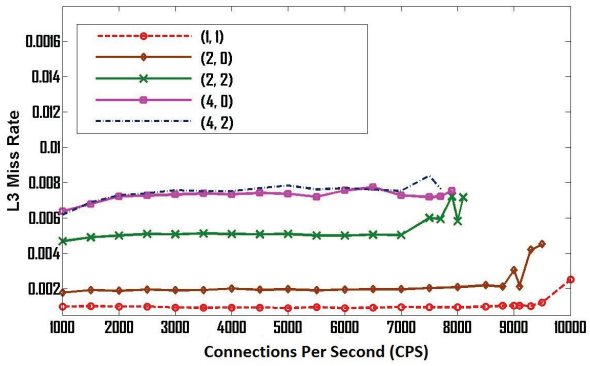

Fig. 3. L3 Miss Rate for Kernel-Intensive Workload

y represent the number of instances in socket 0 and socket 1, respectively. Socket affinity is employed. Figure 2 shows the mean response times of the Apache instances for the kernel-intensive workload. At low loads, response times are not impacted significantly by the number of instances sharing a socket. For example, mean response times are flat up to 5000 cps. However, significant response time degradations occur at higher arrival rates as more and more Apache instances are added to the socket. For example, in Figure 2 the maximum achievable per-instance throughput for the (1, 1) scenario is 10,000 cps whereas it reduces to 9,500 cps for the (2, 0) scenario. Next, we use counter data to try to recognize such a behaviour.

First we consider CPI. The CPI values that correspond to the (2,2) and (4,0) cases of Figure 2 begin at 1.65 for 1000 cps, drop to 1.4 for 4500 cps, and then rise to 1.6 for 8000 cps. This behavior is the same whether we use four cores in the same or different sockets. CPI does not explain response time degradation or even the differences in response times for these two cases. The system does not appear to suffer from a CPI related issue, e.g., contention for processor functional units.

Next, we consider the hardware counter values for L1, L2 and L3 cache miss rates. The L1 and L2 misses were the same in all experiments even when there were considerable differences in performance. However, as shown in Figure 3, the L3 cache misses increase along with the increase in the number of Apache instances in the system for the kernel-intensive workload. Yet, in Figure 2, while the response times of the different cases only start to differ and increase at 5000 cps, the L3 misses for the different cases always differ and do not increase significantly until the maximum cps is achieved. The system incurs different L3 miss rates but does not appear to suffer from an L3 cache contention issue. Similar results were observed for the application intensive workload.

Finally, we find that the CPU utilization value at which response times increase depends on the number of Apache instances and the type of workload. The greater the number of instances the lower the achievable utilization. For 6 instances, the kernel and application intensive workloads can sustain per-core utilizations of 70% and 65%, respectively before experiencing performance degradation. They experience competition for server resources differently so they have different per-core utilization thresholds. As a result using a CPU utilization threshold as an indicator of performance is also problematic. To conclude, these counters do not directly predict or explain the differences in behavior seen in Figure 2.

2) With Virtualization: Our next experiment considers the case where virutalization is employed. For this experiment we ran up to four identical VMs in one socket using socket affinity. Each VM executed an application-intensive micro-benchmark workload. We did not detect any problems for the 1 and 2 VM cases. For 1 and 2 VM cases, we were able to support up to 700 cps per VM, which is only a slight drop from the 750 cps per instance achieved with no-virtualization. However, dramatic performance problems can occur when more VMs are deployed. Figure 4 shows response time results for 4 VMs for increasing cps. Up to a load of 160 cps per VM there is no significant change in the mean response time of VMs. At a load of 165 cps per VM there is a performance discontinuity. In contrast, the mean response time for the similar (4,0) non-virtualized configuration at a load of 165 cps per VM is only around 1.5 ms. When using the VMM with many VMs, the maximum sustainable aggregate connection throughput over all VMs appears to be 660 cps. We note that hardware resources such as the disks and the network were very lightly utilized and that connection establishments to the Apache instances, as observed by the load generators, were taking an inordinate amount of time. This suggested that there were not enough Apache worker processes. However, the maximum number of concurrent connections encountered by any VM was less than the configured Apache limit of 1024. These observations together point to a VMM-level software bottleneck as being responsible for this sudden performance discontinuity. This is another shared server resource that must be managed in virtualized environments.

As with the non-virtualized environment, we assessed whether the commonly used counters are helpful for recognizing this behavior. The CPI value decreases almost linearly from 1.6 to 1.14 over the range of cps shown in Figure 4.

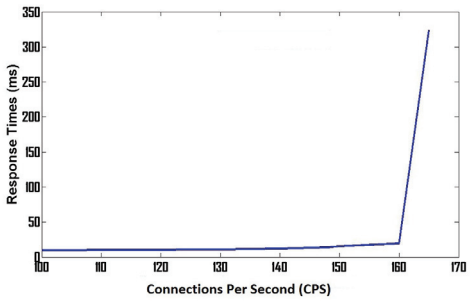

Fig. 4. Performance of 4 VMs in One Socket

TABLE I
VM RESPONSE TIME WITHOUT AND WITH PROBE (WITHOUT, WITH)

CPS	1 VM RT	2 VM RT	3 VM RT	4 VM RT
100	9.2, 9.2	9.5, 9.6	9.6, 9.7	9.7, 10.1
140	11.1, 11.2	11.5, 11.8	11.8, 11.6	12.1, 12.2
155	13.3, 14.0	14.4, 14.5	14.9, 14.8	**15.4, 15.7**
160	14.9, 15.2	15.3, 15.5	15.4, 16.7	**19.3, 18.5**
165	15.2, 15.4	15.8, 15.7	**16.3, 17.4**	**323.6, 1363.8**

This suggests a performance improvement in contrast to the 2000% increase in response time observed. The L3 miss rate varies from $1.9(10^{-3})$ to $1.6(10^{-3})$ over the range of cps but is not strongly correlated with the increasing response times or the performance discontinuity. Finally, the per-core utilization for the 4 VM case at 160 cps and 165 cps were 0.75 and 0.78, respectively. The CPU utilization values do not provide a robust indicator for the performance discontinuity. To summarize, there may be many complex performance issues that are incurred by VMs that may include contention for VMM kernel, OS, and shared hardware resources. A robust method is needed to recognize such contention.

D. A Probe to Predict Contention for Shared Server Resources

This third part of the study employs micro-benchmarks to develop our proposed probe for a virtualized environment. These micro-benchmarks are used to stress server bottlenecks, demonstrate the effectiveness of the probe at recognizing contention for shared server resources, and to tune the probe's parameters. The application intensive Web workload is used as a micro-benchmark that stresses a server's ability to handle high cps. Additionally, we also use the RAMspeed micro-benchmark to stress a socket's caches.

1) Probe for Connections Per Second: We first consider the connections per second phase of the probe to monitor the impact of increasing connections per second. VMs that execute the application-intensive workload are used as micro-benchmarks. Measurement runs were for 100 seconds and were repeated five times so that confidence intervals for response times could be obtained. We consider the connection phase of the probe with c=16 and t_1=100. Since the socket has 4 cores, utilization is between 0 and 4. Table I shows the mean response times in ms (RT as referred to in the table) of the Apache VMs running without and with the probe for various per-VM connection arrival rates and various numbers of VMs. If an application's mean response time falls outside of its 95% confidence interval as compared to the corresponding 1 VM case then a degradation due to contention for shared server resources has occurred. Such results are reported in bold face.

Figure 5 shows the socket's aggregate core utilization for all the cases. From Figure 5, adding VMs to the sockets has an approximately linear effect on the socket's utilization. For example, at 160 cps, the utilization with 4 VMs is about 4 times that of the 1 VM case. However, mean response times exhibit a different trend. From Table I, the addition of a 4th VM causes a modest increase in mean response time with respect to the 1 VM response time for up to 140 cps per VM. For 165 cps per VM, even the addition of a 3rd VM causes a mean response time increase with respect to the corresponding 1 VM case. As described in the previous section, the addition of a 4th VM causes a dramatic performance discontinuity at 165 cps per VM. The socket core utilization data shown in Figure 5 does not capture these trends.

We now consider the overhead of the probe and its effectiveness at recognizing contention for shared server resources. The probe increased the socket's utilization by approximately 0.12, which amounts to increasing the utilization of each core by 3%. Table I also shows the mean VM response times with the probe running as an additional VM. Executing the probe only marginally increases the response times of VMs. The worst case increase in mean response time due to the addition of the probe is 6.7% for the 3 VMs case at 165 cps per VM.

Table II shows the mean response times for the probe. As with the micro-benchmark, cases where the probe undergoes a statistically significant response time degradation are shown in bold. Comparing the values of Table I and Table II, the probe does not capture the relatively subtle increases in VM response times observed for the 4 VMs cases for up to 160 cps and the 3 VMs cases for up to 165 cps. For example from Table I, at 160 cps per VM, the mean response time with 4 VMs is 21% higher than that with 1 VM and this difference is statistically significant – the confidence interval for these two cases do not overlap. The probe's mean response times for both these scenarios is 3.8 ms so it does not recognize this subtle response time degradation. However, the probe is very effective in detecting the sudden performance discontinuity with 4 VMs at 165 cps per VM. The probe's mean response time increases by approximately a factor of 240 when the performance discontinuity occurs. It should also be noted that the probe did not indicate any false negatives, i.e. the probe did not indicate the presence of a performance problem in the VMs when there wasn't one.

2) Probe for Memory Subsystem Contention: Next we consider VMs that execute the RAMSpeed micro-benchmark and the memory phase of the probe. The micro-benchmark executes for 20 minutes. The test is repeated five times so that confidence intervals can be obtained. We progressively increase the number of VMs concurrently executing the benchmark. The probe was run on the socket with n=5000000 and t_2=25. We discuss automated selection of these probe

TABLE II
PROBE RESPONSE TIME FOR APPLICATION-INTENSIVE WEB WORKLOAD

CPS	1 VM RT	2 VM RT	3 VM RT	4 VM RT
100	3.8	3.8	3.9	3.9
140	3.9	3.9	3.9	4.0
155	3.8	3.8	3.9	3.8
160	3.8	3.8	3.9	3.8
165	3.9	3.8	4.0	**924.5**

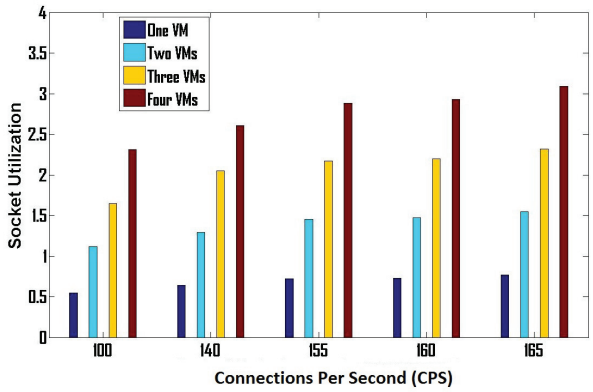

Fig. 5. Socket Core Utilization

parameter values in Section IV-D3. Table III shows the results. We found that the probe introduced negligible overheads. The probe's in-isolation response time was 520 ms with a 95% confidence interval of 520 $\pm$ 6 ms. The probe utilized around 6% of the socket, approximately 1.5% for each core, and caused an L3 miss rate of 0.1 when it ran alone.

From Table III (X= Mean Throughput (MB/s), U= Socket Utilization, MR= Miss Rate), the socket utilization increases almost linearly with every added VM. From the table, the mean throughput of the micro-benchmark degrades for the 2, 3, and 4 VMs cases. As shown in the table, statistically significant degradations in the probe's mean response times were observed for all the cases. Interestingly, the probe's execution time for the 1 VM case is about 5% higher than its in-isolation execution time. The probe VM's response time increases by 63% from 1 VM to 4 VMs, thereby demonstrating good promise for our approach. In contrast to the web-based micro-benchmark results, the probe is able to detect both subtle as well as major performance degradations. It should be noted that the L3 miss rate is a good indicator of performance degradation in this scenario. However, as demonstrated by the web micro-benchmark results, the probe has a more general applicabilty in that it can succeed in scenarios where hardware counters are ineffective.

3) Tuning the Probe: The previous experiments used micro-benchmarks that expose only one type of bottleneck at a time. Furthermore, the probe executed as long as the application VMs. This subsection relaxes these assumptions by automatically tuning the probe's parameters so that the probe alternates between its connection and memory phases.

Specifically, for tuning c and t_1, we set up controlled

TABLE III
PHORONIX MICRO BENCHMARK WITH PROBE

No. of VMs	X	Probe RT	U	L3 MR
1	3393.2	**545**	0.94	0.56
2	3012.1	**630**	1.87	0.63
3	2113.6	**742**	2.82	0.79
4	1803.3	**869**	3.85	0.88

experiments involving the application-intensive web micro-benchmark. Without using the probe, we identify two scenarios with one VM per core, one with an arrival rate per VM that does not have a performance discontinuity and one with an arrival rate that does. We then set c to be a large value, e.g., a cps that when added to the first case will cause the discontinuity. t_1 is set to be the entire duration of the micro-benchmark's execution. The probe is executed for the scenario without the discontinuity. The c value is progressively decreased till the probe does not introduce any performance degradation for the VMs. Next, the probe is run with this setting for the other scenario with the performance discontinuity. We now progressively decrease the value of t_1 up to the point the probe is able to still identify the problem. A similar approach of using the RAMSpeed micro-benchmark for selecting n first followed by t_2 later is adopted for tuning the memory phase of the probe.

Using the above process, we deduce that for the server under consideration the probe settings are c=24, t_1=15, n=5000000, and t_2=25. The connection phases uses 24 cps out of the maximum cps of approximately 600 cps for the socket, which is a monitoring overhead of 4%. The probe when running alone consumed around 5-6% of the entire socket, which is less than 1.5% per core. The in-isolation response time of the connection phase of the probe was 3.7 ms with a 95% CI of 3.7 $\pm$ 0.1 ms. The in-isolation response time of the memory phase of the probe was 525 ms with a 95% CI of 525 $\pm$ 3 ms. We note that the tuned probe runs continuously, alternating phases, with 15 seconds in its connection phase and 25 seconds in its memory phase. It can report to management controllers after each phase.

E. Validation with Realistic Benchmark Workloads

Finally, this fourth part of the study uses the tuned probe to detect problems in scenarios involving benchmark applications that are more representative of real workloads. As mentioned previously, we used the eBay-like RUBiS application as well as memory-intensive Java programs drawn from the DACAPO suite. For RUBiS, we implemented a customized version of a workload generator using httperf. The httperf implementation allowed us to make more efficient use of workload generation resources. It also allowed us to control the connection rate to the server. We used the default browsing mix in all experiments. With DACAPO, we selected the Jython and Antlr programs which have been shown to be memory-intensive by others [2]. Due to space constraints, we only report results for Jython. Jython is a Java tool that interprets the pybench benchmark used for evaluating python implementations.

TABLE V
PERFORMANCE OF MEMORY PHASE OF PROBE FOR JYTHON VMS

No. Of VMs	VM RT (No Probe)	VM RT (With Probe)	Probe RT	U	L3 MR
1	192039	208829	527	0.92	0.23
2	218829	219834	**563**	1.85	0.32
3	234465	234854	**601**	2.82	0.39
4	250819	251432	**721**	3.82	0.46

1) Homogeneous Workloads: First we present results for homogeneous scenarios involving only RUBiS workloads. Similar to the micro-benchmark experiments, VMs serving statistically similar workloads were progressively added to the server. The probe placed negligible overheads on the system. The maximum response time degradation due to the probe was 6%. Each VM executes for 200 seconds. Table IV shows the results of this experiment (We used notations N and Y where N = probe detects no problem and Y = probe detects problem). As done previously, cases where VMs experienced a performance degradation are marked in bold. The VMs start encountering a performance discontinuity when the aggregate connection rate is 600 cps or greater. As with the micro-benchmark workloads, there is a performance discontinuity that affects the VMs after they have executed for approximately 120 seconds and this problem lasts for around 80 seconds. As shown in Table IV, both connection phases of the probe that coincided with this period were able to detect the problem. The response times of the memory phase of the probe were unchanged from the response time when the probe executes in isolation suggesting the absence of any memory-related problems. We note that the results in this case benefited from the last two connection phases occurring after the onset of the discontinuity problems. The entire 15 seconds of these phases coincided with the problem. From our tuning results, it is likely that overlaps less than 15 seconds might cause intermittent transient problems to go undetected.

Table V shows the execution times of a homogeneous scenario containing Jython VMs. As with the RUBiS scenario, there was no significant overhead due to the probe. The connection phases of the probe did not suggest any problems. However, all the memory phases suggested increases to the per-VM execution times with 2, 3, and 4 VMs on the socket. The execution time of Jython increased by 31% from 1 VM to 4 VMs. Based on the execution times of the memory phases of the probe, our approach suggests a performance degradation of 36% for the same change. These results suggest that the tuned probe places negligible overheads while identifying both subtle performance problems and sharp performance discontinuities in homogeneous scenarios.

2) Heterogeneous Workloads: Next we consider three scenarios consisting of both Jython and RUBiS workloads. First we ran 3 RUBiS VMs and 1 Jython VM. We specified two connection rates for the RUBiS VMs, one at which the VMs run without any problem and the other at which the VMs encountered the connection related performance discontinuity problem. Secondly, we ran 3 Jython VMs and 1 RUBiS VM so as to cause only memory problems. Finally, we ran 3 RUBiS and 2 Jython VMs in the socket to have both kinds of problems. The challenge was to see if both kinds of VMs can co-exist in the socket with one another and whether or not the probe could detect the right kind of problems. All VMs ran for 200 seconds along with the probe, which had 5 alternating memory (25 seconds) and connection (15 seconds) phases.

TABLE VI
VM PERFORMANCE FOR 3 RUBIS, 1 JYTHON

CPS	RUBiS VM RT	Jython VM RT	U	L3 MR
180	2.6	201103	2.20	0.31
225	**1045.3**	201056	2.32	0.32

TABLE VII
VM PERFORMANCE FOR 3 JYTHON, 1 RUBIS

CPS	RUBiS VM RT	Jython VM RT	U	L3 MR
180	2.6	**235130**	3.40	0.58
225	2.8	**235661**	3.46	0.62

For the first case, the RUBiS VMs ran at two rates, 180 and 225 cps respectively. At 180 cps there was no performance problem. At 225 cps, however, the VMs ran into the performance discontinuity problem as observed earlier. The results are illustrated in Table VI. The Jython VM ran without encountering any issues. The same experiment was then executed along with the probe VM running in the socket. Again, the probe did not cause much impact to VM performance. The memory phases of the probe did not indicate any problem but the latter connection phases of the probe were able to detect the performance discontinuity problem for the 225 cps per VM case (Table IX).

The second experiment was conducted with 3 Jython VMs and 1 RUBiS VM. The Jython VMs ran into memory-related problems and these were identified by the memory phases of the probe. The connection phases of the probe did not suggest any problem. The results for performance of the VMs and the probe are shown in Tables VII and IX respectively. A final experiment was done with 3 RUBiS VMs and 2 Jython VMs. This experiment was designed so as to run into both kinds of problems concurrently. The two phases of the probe reported both the problems when they occurred as seen in Tables VIII and IX.

V. BACKGROUND AND RELATED WORK

A significant body of research exists on managing application performance in consolidated environments [9], [4], [10], [12], [1]. For example, application-level approaches have been proposed where application response time measures are used

TABLE VIII
VM PERFORMANCE FOR 3 RUBIS, 2 JYTHON

CPS	RUBiS VM RT	Jython VM RT	U	L3 MR
180	2.8	**220866**	3.19	0.63
225	**988.6**	**221042**	3.23	0.68

TABLE IV
PERFORMANCE OF CONNECTION PHASE OF PROBE FOR RUBIS VMS

CPS	1 VM RT	2 VM RT	3 VM RT	4 VM RT	1 VM-Probe	2 VM-Probe	3 VM-Probe	4 VM-Probe
100	1.5	1.5	1.6	1.8	N	N	N	N
125	1.6	1.6	1.8	2.0	N	N	N	N
150	1.8	2.1	2.2	2.3	N	N	N	N
180	2.0	2.2	**2.5**	**226.4**	N	N	N	Y
225	2.3	2.5	**1045.3**	**2243.1**	N	N	Y	Y

TABLE IX
PROBE PERFORMANCE FOR HETEROGENEOUS WORKLOADS

CPS	Phase1 (Mem)	Phase1 (I/O)	Phase2 (Mem)	Phase2 (I/O)	Phase3 (Mem)	Phase3 (I/O)	Phase4 (Mem)	Phase4 (I/O)	Phase5 (Mem)	Phase5 (I/O)
3 RUBiS, 1 Jython										
180	N	N	N	N	N	N	N	N	N	N
225	N	N	N	N	N	N	N	Y	N	Y
3 Jython, 1 RUBiS										
180	Y	N	Y	N	Y	N	Y	N	Y	N
225	Y	N	Y	N	Y	N	Y	N	Y	N
3 RUBiS, 2 Jython										
180	Y	N	Y	N	Y	N	Y	N	Y	N
225	Y	N	Y	N	Y	N	Y	Y	Y	Y

at runtime to initiate requests to obtain or relinquish VMs from a virtualized resource pool in response to workload fluctuations [9], [4], [13]. However, we want to infer the impact of the interference due to competition for unmanaged server resources upon multiple application VMs. Our problem arises at shorter time scales where such application measures do not provide sufficient information. Several studies exist that consider detecting at runtime the impact of contention among shared batch workloads [5], [6], [3], [14], [15] and enterprise workloads with limited concurrency, i.e., where number of software threads is less than or equal to the number of cores in a host [6], [3]. In contrast, Kousiouris et al. develop an offline, pre-deployment approach that relies on an artificial neural network model to decide whether a given set of applications can be consolidated on the same server [10]. All of these approaches rely on metrics such as CPU utilization, CPI and cache miss rates to detect adverse performance. We have shown these measures are not always effective for interactive Web server workloads that are typical for cloud computing environments. Furthermore, these studies only considered contention for hardware resources and did not focus on VMM-level software bottlenecks.

VI. SUMMARY AND CONCLUSIONS

The probe approach permits an operator of a virtualized resource pool to detect contention for shared server resources such as VMM software resources and memory hierarchies at runtime without relying solely on VMM and hardware performance counters. Hardware counters do not always capture the impact of shared resource contention on all types of workloads. Moreover, hardware counters are difficult to use as their availability and usage model can change between architectural revisions from the same manufacturer and greatly differ between the architectures of different manufacturers. Our position is that a probe is a significantly more portable and lightweight approach that provides direct feedback on contention, regardless of the root cause, and that it is sufficient to recognize when a host can no longer sustain the variety of workloads in VMs assigned to it. We proposed a tuning approach that can reduce the effort to deploy the probe in different environments.

Our approach is complementary to workload placement and other management control systems. The probe approach focuses on recognizing contention for shared resources that can affect application performance in complex ways. If a problem arises then a management system can be notified. It can then initiate the migration of a VM from one socket to another or to another host to reduce the impact of the contention on applications. To properly employ our method thresholds are still needed to decide whether a problem exists. The connection and memory phases of the probe rely on different kinds of thresholds. A cloud provider may work with customers to determine that the memory sensor's response time should not be, say, more than p percent higher than when run in isolation, as a measure of what level of cache interference is acceptable.

Future work can focus on studying how our results generalize to other types of servers, applications, and virtualization software. We repeated a subset of experiments on an Intel Xeon E5620 server and observed similar results, which indicates good promise for our approach. We note that since the probe runs on each socket in a resource pool, our approach can also work for multi-tier applications involving many interacting VMs. Our next steps include deploying the approach in a cloud environment to determine how often it recognizes contention for shared server resources and then integrating it with management systems to overcome such problems.

REFERENCES

[1] Lydia Y. Chen, Danilo Ansaloni, Evgenia Smirni, Akira Yokokawa, and Walter Binder. Achieving application-centric performance targets via consolidation on multicores: myth or reality? In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12, pages 37–48, New York, NY, USA, 2012. ACM.

[2] Huang Xianglong et al. The garbage collection advantage: improving program locality. In *Proc. of OOPSLA '04*, pages 69–80. ACM, 2004.

[3] Lingjia Tang et al. The impact of memory subsystem resource sharing on datacenter applications. *SIGARCH Comput. Archit. News*, 39(3):283–294, June 2011.

[4] Rahul Singh et al. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. of ICAC'10*, pages 21–30. ACM, 2010.

[5] Ramesh Illikkal et al. Pirate: Qos and performance management in cmp architectures. *SIGMETRICS Perform. Eval. Rev.*, 37(4):3–10, March 2010.

[6] Sergey Blagodurov et al. Contention-aware scheduling on multicore systems. *ACM Trans. Comput. Syst.*, 28(4):8:1–8:45, December 2010.

[7] Stephen M. Blackburn et al. The dacapo benchmarks: java benchmarking development and analysis. *SIGPLAN Not.*, 41(10):169–190, October 2006.

[8] Todd Deshane et al. Quantitative comparison of xen and KVM. In *Xen summit*, Berkeley, CA, USA, June 2008. USENIX association.

[9] Waheed Iqbal et al. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871 – 879, 2011.

[10] George Kousiouris, Tommaso Cucinotta, and Theodora Varvarigou. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *J. Syst. Softw.*, 84(8):1270–1291, August 2011.

[11] David Mosberger and Tai Jin. httperf a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26(3):31–37, December 1998.

[12] Bryan Veal and Annie Foong. Performance scalability of a multi-core web server. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, ANCS '07, pages 57–66, New York, NY, USA, 2007. ACM.

[13] Zhikui Wang, Yuan Chen, D. Gmach, S. Singhal, B.J. Watson, W. Rivera, Xiaoyun Zhu, and C.D. Hyser. Appraise: application-level performance management in virtualized server environments. *Network and Service Management, IEEE Transactions on*, 6(4):240 –254, december 2009.

[14] Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proc. of ICAC'11*, pages 225–234. ACM, 2011.

[15] Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 225–234, New York, NY, USA, 2011. ACM.