The Impact of Software Resource Allocation on Consolidated n-Tier Applications

Jack Li*, Qingyang Wang*, Chien-An Lai*, Junhee Park*, Daisaku Yokoyama†, Calton Pu*
*Center of Experimental Computer Science, Georgia Institute of Technology, Atlanta, GA, USA
{jack.li, qywang, clai31, jhpark, calton}@cc.gatech.edu
†Institute of Industrial Science, University of Tokyo, Tokyo, Japan
yokoyama@tkl.iis.u-tokyo.ac.jp

***Abstract*—Consolidating several under-utilized user applications together to achieve higher utilization of hardware resources is important for cloud vendors to reduce cost and maximize profit. In this paper, we study the impact of tuning software resources (e.g., server thread pool size or connection pool size) on n-tier web application performance in a consolidated cloud environment. By measuring CPU utilizations and performance of two consolidated n-tier web application benchmark systems running RUBBoS, we found significant differences depending on the amount of soft resources allocated. When the two systems have different soft resource allocations and are fully utilized, the application with more software resources may steal up to 8% CPU from the co-resident application. Further analysis shows that the CPU stealing is due to more threads being scheduled for the system with higher software resources. By limiting the number of runnable active threads for the consolidated VMs, we were able to mitigate the performance interference. More generally, our results show that careful software resource allocation is a significant factor when deploying and tuning n-tier application performance in clouds.**

***Keywords*-application co-location; consolidation; software resources; n-tier; performance; RUBBoS; sharing.**

I. Introduction

Hardware virtualization technologies have been a popular technique for cloud vendors to allow multiple applications to be consolidated into a single server, increasing the utilization of shared cloud infrastructure. Although consolidation is simple in concept, significant practical complications have been found, e.g., the noisy neighbor problem [1], [2] where co-located applications interfere with each other. In response to noisy neighbor problem, one might assume that perfect isolation is the solution of choice for consolidated applications. Instead, we have found that *sharing is better than isolation* [3], where co-located applications achieve better overall performance through sharing of CPU compared to perfect isolation that limited the utilization of a critical resource.

In this paper, we explore the sharing option in consolidation, since it provides a better chance of achieving higher overall utilization. Our experimental environment reflects typical production mission-critical web-facing n-tier applications. Specifically, we are interested in the measuring mutual influence among well-behaved neighbors when they make very reasonable choices in software resource (e.g., thread/DB connection pool of a web or application server) allocation. In previous research [4], we have found that performance of an n-tier application is significantly affected by the allocation of both hardware and software resources. Experimental evidence showed that changing only the software resource allocation of an application can degrade throughput by up to 90%.

The paper's main contribution is an in-depth experimental study analyzing soft resource allocation's impact on the performance of two consolidated n-tier applications. Specifically, we run the RUBBoS benchmark [5] on the KVM hypervisor. We show non-trivial performance differences caused by different settings of soft resource allocations at high system workloads (the area of most interest). First, if both consolidated n-tier applications are set to a low allocation of soft resources, they both achieve the best performance in terms of goodput and response time. Second, when both consolidated systems are set to a high allocation of soft resources, the last level cache miss overhead greatly degrades the performance of both systems as workload increases. Third, when one system is set to high soft resource allocations and the other is set to low allocations, the system with higher allocation can "steal" as much as 8% of CPU from the system with a lower setting of soft resources. Although we focus on RUBBoS running on KVM, these performance differences can occur in other n-tier applications and hypervisors.

The paper's second contribution is a practical solution to resolve the CPU stealing problem by limiting the number of threads that can spawn in each VM process. By default, the KVM hypervisor allows its virtual machines to spawn any number of threads. When concurrency is high within a VM due to a high allocation of soft resources, the virtual machine operating system will spawn more threads. This causes more threads to be created at the hypervisor level for the virtual machine. KVM uses Linux's Completely Fair Scheduler (CFS) which treats each thread as equal during scheduling. Thus, when there is a disparity in the number of threads between VMs, the VM with the greater number of threads will be scheduled more and steal CPU from the VM with less threads. When we limit the number of threads for each virtual machine process within KVM, it prevents any one VM from spawning too many threads and stealing CPU from other VMs.

The remainder of this paper is structured as follows. Section II gives background information and summarizes related work. Section III provides an overview of the experimental setup. In Section IV we study the performance of two

978-1-4799-5063-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CLOUD.2014.51

consolidated n-tier applications with the same soft resource allocation while in Section V we study the performance of these two applications when the soft resource allocations for each are different. Section VI discusses an approach to solving the CPU "stealing" problem through limiting threads. Finally, Section VII concludes the paper.

II. Background & Related Work

The impact of soft resource allocations on system performance has been previously studied in [4], [6], [7]. Wang et al. [4] showed how different allocations of software resources had drastic impacts on n-tier application performance. The authors illustrated that setting an appropriate balance of soft resources could increase total goodput and decrease response time in the system. Our work introduces the impact of soft resource allocation on two n-tier applications competing for CPU resources.

Previous researchers explored different application placement strategies on cloud environments [8]–[10], but most works assume linear consolidation performance and use bin-packing techniques for VM placement [11], [12]. Ferreto et al. [13] applied linear programming to dynamic VM migration to improve consolidated application performance. Our work shows that soft resource allocation is also an important factor when considering the deployment strategy for consolidated n-tier applications.

The interference effect from application consolidation in cloud environments has also been studied ([14]–[16]). At high resource utilizations, application consolidation can cause the noisy neighbor problem. Xing et al. [2] studied the performance factors that can impact resource multiplexing and scheduling among virtual machines and found many sources of I/O and CPU contention between co-resident VMs. Koh et al. [1] showed that different types of workloads may cause hidden contention for physical resources and cause performance interference between colocated VMS. Malkowski et al. [17] introduced the non-monotonic response time phenomenon in consolidated n-tier applications which showed that even in isolated environments where a hard limit of resources was set for VMs, interference between virtual machines on the same physical node could cause significant performance degradations. Kanemasa et al. [3] showed that when VMs shared the same physical node, setting those VMs to share 100% of the CPU was better than having isolated VMs with limited, but protected resources. Our work corroborates the interference found in these past works, but in addition, we also show that even in a 100% sharing scenario, one n-tier application can gain an unfair share of CPU utilization by changing the soft resource allocation of the application.

III. Experimental Setup

Although consolidation may be used for any type of application, our paper focuses on the consolidation of n-tier applications with LAMP (Linux, Apache, MySQL and PHP) implementations. N-tier application architecture is a pipeline of separate applications with each application serving a distinct

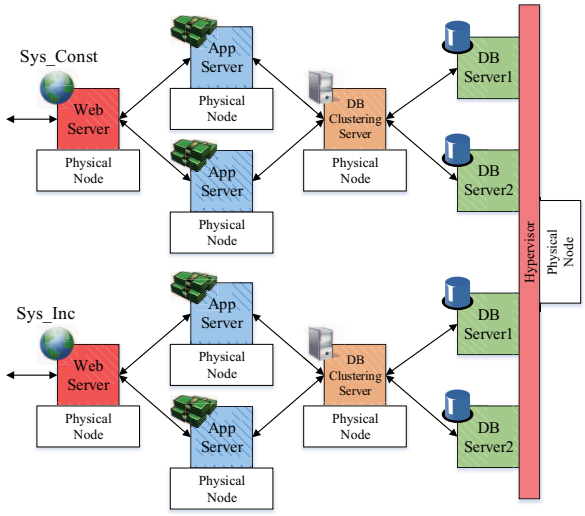
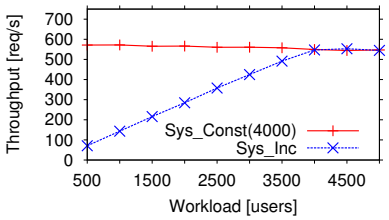


Fig. 2: System deployment of our experiment. The first three software servers are on dedicated nodes. The DB servers of Sys_Const and Sys_Inc are dedicated VMs on a single shared physical node. DB Server 1 of Sys_Const and Sys_Inc share the same physical core while DB Server 2 of both systems share another core.

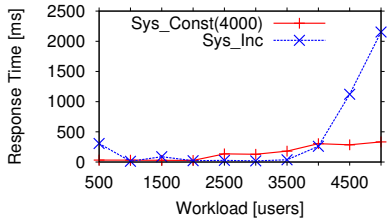
role. A standard 3-tier application will consist of a web server (e.g., Apache), an application server (e.g., Tomcat), and a database server (e.g., MySQL). In our experiments, we have an additional database clustering middleware server in between the application and database tiers since we use more than one database server. In our experiments, we use the popular n-tier application benchmark system RUBBoS.

Although any server can be consolidated into a physical node, we focus on consolidating only database servers together into a single physical node using the KVM hypervisor. KVM is part of the Linux operating system and uses the default Linux CPU scheduler and memory management for fully virtualized guest VMs supported by processor extensions. Figure 2 shows our two system deployment each with one web server, two application servers, one database clustering middleware, and two database servers (denoted 1-2-1-2). In our experiments, we denote the first RUBBoS system as Sys_Const and the second RUBBoS system as Sys_Inc. Sys_Const's workload is always at 4,000 users while Sys_Inc's workload varies from 500 users to 5,000 users. The consolidation methodology for our experiments is to give the web servers, application servers, and database clustering middleware its own physical node. We then consolidate all four MySQL database servers into one physical node with each MySQL server being hosted on its own dedicated VM. The vCPU of the first MySQL server from Sys_Const and Sys_Inc is pinned to one physical CPU and the vCPU of the second MySQL server from both systems is pinned to another physical CPU on a separate package. By default, the VMs are fully sharing the physical core (100% sharing). The hardware specifications for the physical node and virtual machine can be found in Table I.

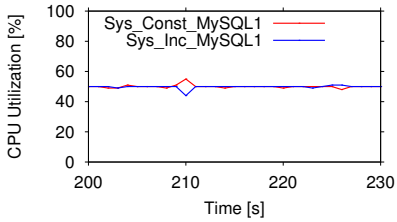
In this paper, we explore how software resource alloca-



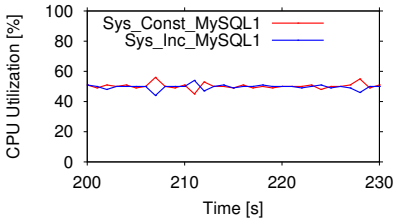
(a) Throughput when both systems have Low-SR (LL-SR) allocations. Sys_Const's throughput remains constant while Sys_Inc's throughput gradually increases and peaks when the workload of the two systems are the same.



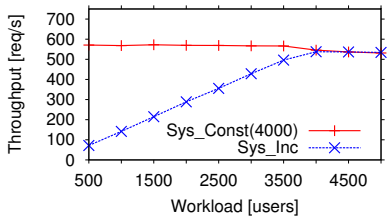
(b) Response time when both systems have Low-SR (LL-SR) allocations. Sys_Inc's response times spike due to not having enough resources to support the workload.



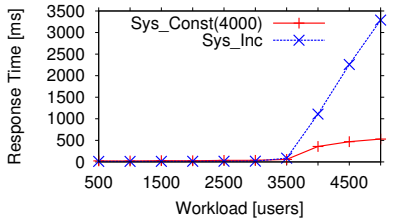
(c) CPU utilization of Sys_Const and Sys_Inc seen at the hypervisor level when the workload of Sys_Inc is 4,000. The graph shows VM isolation as the VMs are all sharing 50% of its allotted core.



(d) CPU utilization of Sys_Const and Sys_Inc for Sys_Inc workload 5,000. The hypervisor still maintains a 50/50 split of CPU even when Sys_Inc's workload is greater than Sys_Const.



(e) Throughput when both systems have High-SR (HH-SR) allocations. The trend is similar to the Low-SR case shown in Figure 1(a).



(f) Response time when both systems have High-SR (HH-SR) allocations. The trend is similar to the Low-SR case shown in Figure 1(b).

Fig. 1: Performance comparison when both systems have the a Low-SR allocation. The throughput for Sys_Inc increases and then peaks at workload 4,000 to become the same as Sys_Const. The response time for both systems is low until Sys_Inc's workload exceeds Sys_Const's workload. Figure 1(c) and 1(d) shows that each VM is using 50% CPU of a core which shows fair sharing even when Sys_Inc's workload is higher than Sys_Const's workload. Figure 1(e) and 1(f) shows the throughput and response times when both systems have a High-SR allocation which shows similar performance to the LL-SR case.

TABLE I: Hardware and VM Configuration.

Physical Machine	
Processor	2 X Intel(R) Xeon(R) @ 2.40GHz (Quad)
Memory	24GB
Operating System	Debian Linux 6.0.7 (squeeze)
Virtual Machine	
Virtual CPUs	1
Memory	2GB
Disk Space	25GB
Operating System	Red Hat Enterprise 6

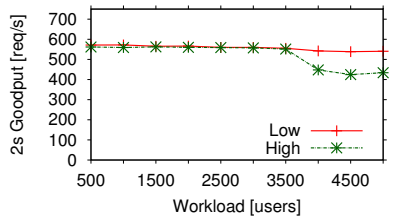
TABLE II: Configurations of major software resources

Tier	Parameter name	Configuration name	
		High-SR	**Low-SR**
Apache	MaxClients	340	120
	ThreadsPerChild	170	60
	WorkerConnection PoolSize	100	25
Tomcat	maxThreads	240	60
DB connections	total	96	16
	each Servlet	12	2

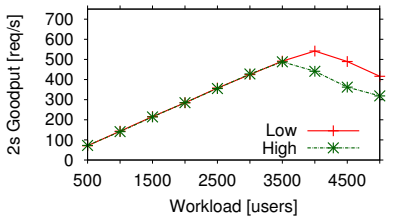
tion can impact the performance when consolidating n-tier applications. Soft resources are parameters that can be tuned within each application such as Tomcat's thread pool or DB connection pool size. We use two different allocations of soft resources as shown in Table II: a low allocation (Low-SR) and a high allocation (High-SR).

IV. Same Software Resource Allocations

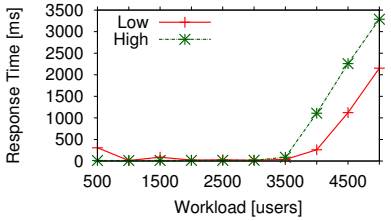
In our first set of experiments, we set both Sys_Const and Sys_Inc to have the same soft resource allocation. We set both systems to have a Low-SR allocation (LL-SR) and both systems to have a High-SR allocation (HH-SR). The parameters for both the Low-SR and High-SR allocations can be found in Table II. Figure 1(a)-(b) shows the throughput and response time of Sys_Const and Sys_Inc when both systems have a Low-SR allocation (LL-SR). As shown in Figure 1(a), Sys_Const's throughput stays the same as the workload of Sys_Inc increases while Sys_Inc's throughput increases until it reaches workload 4,000 and then levels off. The bottleneck in both systems is the database CPU (due to space constraints we do not show the CPU utilizations). This behavior is expected since the VMs of the two systems should share an equal amount of CPU (50/50) when the core is being fully utilized. Figure 1(b) shows the response time of the two systems. Sys_Const's response time is steadily increasing as the workload of Sys_Inc increases. This shows that even at low workloads, Sys_Inc is affecting the application performance of Sys_Const. On the other hand, Sys_Inc's response time seems



(a) 2s Goodput of Sys_Const under LL-SR and HH-SR allocations. When both systems have the same SR allocation, the Low-SR allocation achieves higher goodput at high workloads.



(b) 2s Goodput of Sys_Inc under LL-SR and HH-SR allocations. Similar to the Sys_Const case, the Low-SR allocation again achieves better throughput at high workloads.



(c) Response Time of Sys_Inc under LL-SR and HH-SR allocations. When both systems have the same SR allocation, the Low-SR allocation achieves better response times at high workloads.

Fig. 3: Goodput and response time when both systems have the same software resource allocation. At high workloads (high resource utilizations), the low soft resource allocations allow both systems to have higher goodput and lower response time. The goodput analysis reveals hidden performance problems that is masked by looking only at throughput without considering SLA.

to stay the same until it reaches workload 4,000 and then the response time drastically increases. This is due to Sys_Inc's higher workload relative to Sys_Const. Sys_Inc wants to grab more resources but is unable to do so, so overall performance suffers as a result in terms of response time even though throughput shows a constant behavior. Figure 1(c) shows the CPU utilization of the first MySQL VM server for Sys_Const and Sys_Inc. We show only the first MySQL server of both systems since the behavior of the second MySQL server shows similar behavior.From the figure we see that the MySQL VMs are sharing half the core each. Similarly, Figure 1(d) shows the CPU utilization of the MySQL VMs at Sys_Inc workload 5,000. Even when the workload of Sys_Inc is higher than the workload of Sys_Const, the hypervisor is able to maintain fairness in CPU scheduling between the two systems.

Figure 1(e)-(f) shows the throughput and response time of Sys_Const and Sys_Inc when the two systems have a High-SR allocation (HH-SR). The throughput and response time behavior is similar to the LL-SR allocation case; however, in previous work [7], we showed in addition to affecting throughput and response time, soft resource allocation also plays a major impact in affecting goodput. Goodput is related to throughput, but we only count a request as completed if it falls within a minimum response time threshold. In typical n-tier applications, business providers consider a threshold of 500ms to be an acceptable response time for a request. In this paper, we will be more lenient and consider requests with less than 2 seconds as acceptable. Figure 3(a)-(b) shows the 2s goodput of Sys_Const and Sys_Inc respectively for the same LL-SR and HH-SR allocations. The goodput analysis shows a hidden performance difference that was previously masked by only looking at throughput without considering SLA. The two systems have the highest goodput when they both have a Low-SR allocation. Response times for the two systems are also lower at the Low-SR allocation as shown in Figure 3(c) (we only show the response time of Sys_Inc due to space constraints).

This result is contrary to previous results found when consolidating two n-tier applications together ([3], [17]) which showed that setting soft resources to a high value would

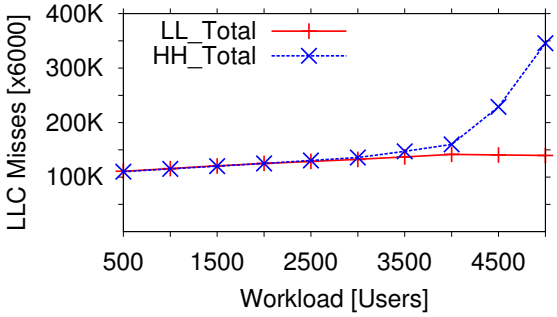
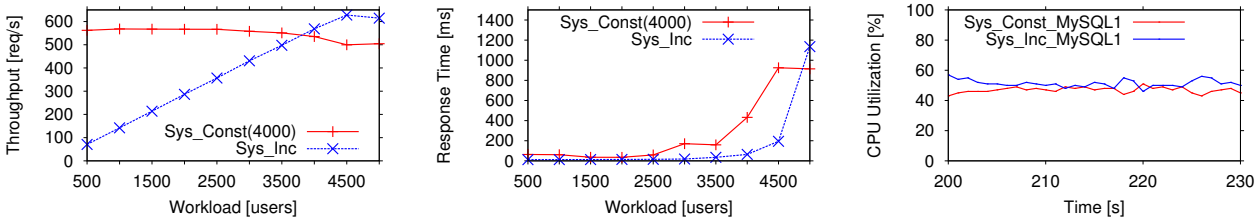


Fig. 4: Total last level cache misses for both systems under the LL-SR and HH-SR allocations. When both systems have a Low-SR, the last level cache misses for each core remain constant regardless of the workload. When both systems have a High-SR, the last level cache misses drastically increases when the workload is greater than 4,000.

improve the non-monotonic response time phenomenon due to higher system concurrency and a better utilization of system resources. Our system configuration differs from the previous works in that our system has one more web and database servers as well as an additional C-JDBC server for load balancing between the two MySQL servers. We found that with this additional MySQL server, the higher concurrency due to high soft resource allocation greatly increased the number of L3 cache misses in the system. Figure 4 shows the last-level-cache misses for the two cores under both the LL-SR and HH-SR configurations. The number of L3 cache misses stays relatively stable from workload 1,000-3,000, but drastically increases once the workload reaches 4,000 for Sys_Const. The total L3 cache misses at workload 5,000 is almost threefold for the HH-SR allocation than the LL-SR allocation.

V. Different Software Resource Allocations

Section IV showed that consolidated systems with Low-SR allocations achieves better performance than consolidated systems with High-SR allocations due to the greater number of L3 cache misses at high workloads. In this section, we set the soft resource allocation for Sys_Const to be Low-SR and for Sys_Const to be High-SR (LH-SR).

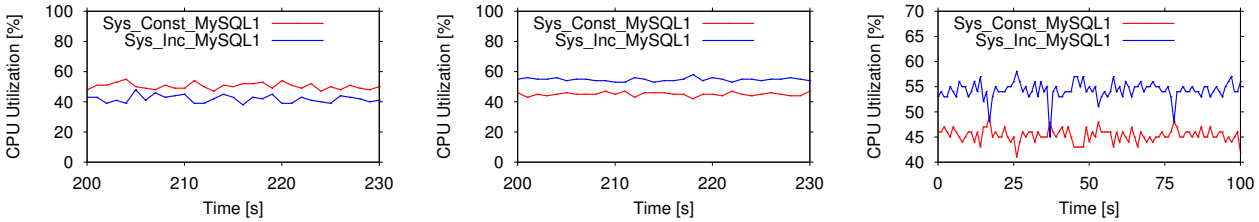


(a) Throughput for LH-SR allocation. Even when the workloads are the same, Sys_Inc's throughput is 15% higher than Sys_Const's throughput.

(b) Response time for LH-SR allocation. Sys_Inc's response time is lower at all workloads than Sys_Const's response time except for 5,000.

(c) CPU util. at the hypervisor level of the both systems' MySQL1 VM at Sys_Inc workload 4,000. Sys_Inc can steal up to 8% CPU from Sys_Const.

Fig. 5: Throughput and response times when Sys_Const has a Low-SR allocation and Sys_Inc has a High-SR allocation (LH-SR). As opposed to when both systems had the same soft resource allocation, Sys_Inc now has higher throughput and also lower response time at high workloads caused by CPU "stealing". Figure 5(c) shows the CPU utilization at the hypervisor for each database VM when the workload of Sys_Inc is 4,000 (same as Sys_Const's workload). The VMs of Sys_Inc are getting more CPU than Sys_Const's VMs.



(a) CPU Utilization at the hypervisor level of each MySQL VM for LH-SR allocation. The workload of Sys_Inc is 3,000. The VMs of Sys_Const are using more CPU than the VMs of Sys_Inc.

(b) CPU Utilization at the hypervisor level of each MySQL VM for LH-SR allocation. The workload of Sys_Inc is 5,000. The VMs of Sys_Inc are able to get more CPU than the VMs of Sys_Const.

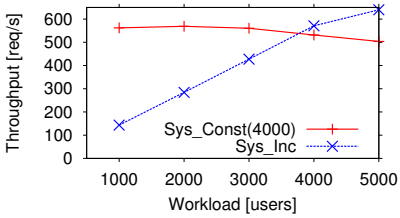
(c) A 100 second trace of CPU utilization of MySQL1 for Sys_Const and Sys_Inc under a LH-SR allocation. The workload of Sys_Inc is 5,000. Dips in the CPU utilization of one VM is immediately snatched up by the other VM.

Fig. 6: CPU Utilization at the hypervisor level of each MySQL VM for LH-SR allocation. As the workload of Sys_Inc increases, the CPU it is allotted also increases. As opposed to when both systems have the same soft resource allocation, in this scenario, Sys_Inc is actually able to get more CPU for its VMs from the hypervisor.

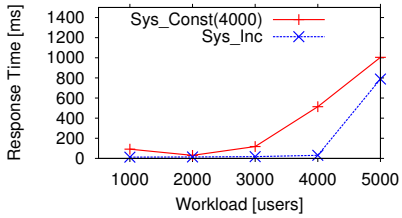
Based on the results from Section IV, we suspect that in this configuration Sys_Const will achieve better performance than Sys_Inc due to the lower concurrency of the system and thus lower L3 cache misses. Figure 5 shows the throughput and response time for the LH-SR allocation. We see in Figure 5(a) that Sys_Inc's throughput is actually able to surpass Sys_Const's throughput at high workloads of Sys_Inc which contradicts our initial assumption. In fact, even when the workloads for both systems are the same (4,000 users), Sys_Inc is able to achieve 6% more throughput than Sys_Const. The throughput of Sys_Const slowly degrades at high workloads of Sys_Inc and actually falls 12.6% of its max value. Again, the bottleneck of the system is the CPU at the database tier. Figure 5(b) shows the response time for the LH-SR allocation. The response time for Sys_Inc is lower at all workloads than Sys_Const's response time, except for the highest workload of 5,000 users. When the two systems have the same workload, the response time for Sys_Const is almost seven times greater than Sys_Inc's response time. The higher soft resource allocation of Sys_Inc is able to achieve higher throughput and lower response time than Sys_Const. Figure 5(c) shows a 30 second trace of each database VM's CPU utilization as seen by the hypervisor. The average CPU utilization of Sys_Const's VMs during this time period is 43.66% and the average CPU utilization for Sys_Inc's VMs is 55.9%. From this timeline, we see that Sys_Inc can steal as much as 8% CPU from Sys_Const.

Figure 6(a) and 6(b) shows the CPU utilizations of the database VMs at Sys_Inc workloads 3,000 and 5,000, respectively. When Sys_Inc's workload is 3,000, we see that Sys_Const's VMs are always using more CPU than Sys_Inc's VMs. This is expected since the workload for Sys_Const is higher than the workload of Sys_Inc; however, when Sys_Inc's workload is 5,000, Sys_Inc's VMs are always getting more CPU than Sys_Const's VMs as shown in Figure 6(b). Figure 6(c) shows a 100 second trace of the same experiment where CPU "stealing" is occurring. The figure shows the CPU utilization for Sys_Const and Sys_Inc's MySQL1 VM which shares the same physical CPU. Again, we can see that Sys_Inc's MySQL VM can snatch up to 8% CPU from Sys_Inc's MySQL VM.

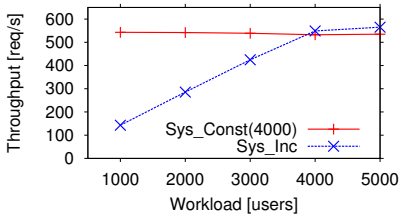
We diagnosed the CPU "stealing" problem to a difference between the number of MySQL DB connections per servlet in the database tier and to the maxClients parameter for Tomcat in the application tier. The higher soft resource allocation results in higher concurrency in the system which allows it to



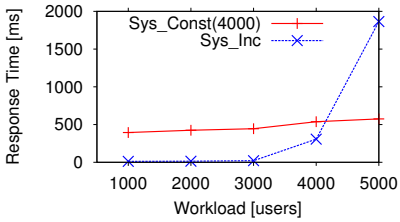
(a) Throughput of LL-SR allocation with DB connections (DBX) for Sys_Const set to 2 and DB connections for Sys_Inc set to 4. Doubling the number of database connections at low concurrency levels greatly increases throughput.



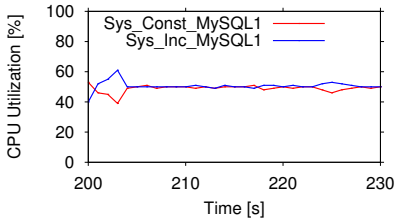
(b) Response time of LL-SR allocation with number of with DBX=2 for Sys_Const and DBX=4 for Sys_Inc. The response time for Sys_Inc is lower than Sys_Const's response time at all workloads.



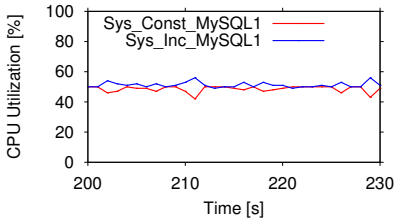
(c) Throughput of HH-SR when Sys_Const's Tomcat maxClients=60 and Sys_Inc's Tomcat maxClients=240. Even when concurrency is high for both systems, a slight change in soft resource allocations can yield significant benefits.



(d) Response time of HH-SR when Sys_Const's Tomcat maxClients=60 and Sys_Inc's Tomcat maxClients=240. The response time is lower for Sys_Inc for all workloads except for 5,000.



(e) CPU utilization at the hypervisor level of the VMs of HH-SR allocation when Sys_Const's Tomcat maxClients=60 and Sys_Inc's Tomcat maxClients=240. The workload of Sys_Inc is 4,000 (same as Sys_Const).



(f) CPU utilization at the hypervisor level of the VMs of HH configuration when number of Tomcat connections for Sys_Const is set to to 60 and the number of Tomcat connections for Sys_Inc is set to 240. The workload of Sys_Inc is 5,000.

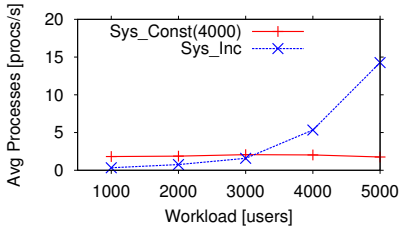
Fig. 7: Discrepancy in throughput and response time between Sys_Const and Sys_Inc is mainly due to the different allocations in the number of DB connections per RUBBoS servlet and in the number of Tomcat maxClient connections accessing those DB connections. Even when concurrency is low for both systems as shown in Figure 7(a) and 7(b), a slight increase in the DB connections per servlet can greatly affect throughput and response time. Likewise, when the concurrency is sufficiently high for both systems as shown in Figure 7(c) and 7(d), a change Tomcat maxClients also affects performance although not as significantly as changing the DB connections parameter.

obtain extra CPU. Figure 7(a) and 7(b) shows the throughput and response time of an LL-SR allocation but we change the number of DB connections per servlet for Sys_Inc to be 4 (instead of 2). When we only change the database connections per servlet, Sys_Inc is able to obtain higher throughput and better response times which is similar to the LH-SR case. At Sys_Inc workload=4,000, we see that the throughput difference between the two systems is only 3%, slightly lower than the LH-SR case. The response time behavior is also similar to the LH-SR case with Sys_Inc achieving lower response times than Sys_Const at all workloads. Figure 7(c) and 7(d) shows the throughput and response times of an HH-SR allocation but we change the number of maxClients for Sys_Inc to be 240 (instead of 60). We see that at workloads 4,000 and 5,000 for Sys_Inc, the throughput is still higher than Sys_Const's workload; however, the difference is not as drastic as changing the database connections. Sys_Const's response time is much higher than in previous cases at 400 ms, but it remains relatively stable regardless of Sys_Inc's workload.

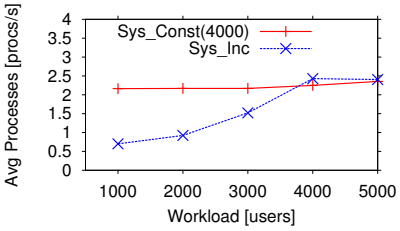
VI. Setting the Limit on Threads per VM Process

From Section V, we know that the CPU "stealing" problem is caused by a difference the allocation of Tomcat maxClients and the database connection pool size. When a system has higher concurrency at a particular tier (e.g., database tier), it causes the VMs at those tiers to have a higher number of active processes running on the VM. For the KVM hypervisor, the number threads running inside the VM directly correlates to the number of active threads running for the VM at the hypervisor level. This means the higher concurrency system will have more active threads in the hypervisor than the lower concurrency system when the soft resource allocations are different. The default KVM scheduling policy for the CPU of its VMs is to adopt the Linux kernel's Completely Fair Scheduler (CFS). CFS treats each thread as equal and tries to schedule threads so that each one gets a similar amount of runtime. Since the High-SR system has more active threads than the Low-SR system, the KVM hypervisor will schedule more of High-SR system's threads.

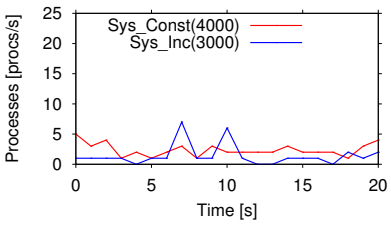
Figure 8(a) and 8(b) shows the average active processes per second for Sys_Const and Sys_Inc under LH-SR and HH-SR allocations, respectively. When the soft resource allocations of the systems are different, we see that Sys_Inc has more active processes per second than Sys_Const once the workload reaches 4,000. When the soft resources are the same for both systems, Sys_Inc's active processes is lower than Sys_Const's active processes, but peaks when the workloads of the two systems are the same. Figures 8(c), 8(d), 8(e) show a 20 second time line trace of Sys_Const and Sys_Inc's active processes per second for Sys_Inc workloads 3,000, 4,000 and



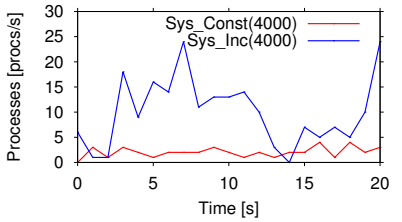
(a) Avg. processes per second being run for the database VMs at the hypervisor level under the LH-SR allocation. At high workloads, Sys_Inc has more active processes than Sys_Const.



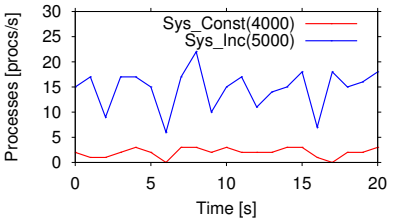
(b) Avg. processes per second being run for the database VMs under the HH-SR allocation. When the two systems have the same allocation, the active processes are the same for both systems.



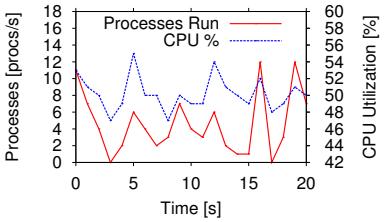
(c) 20 second time line of processes run/s for Sys_Const and Sys_Inc at Sys_Inc workload=3,000 under a LH-SR allocation. On average, Sys_Const has more processes run than Sys_Inc.



(d) 20 second time line of processes run/s for Sys_Const and Sys_Inc at Sys_Inc workload=4,000 under a LH-SR allocation. On average, Sys_Inc has more processes run than Sys_Const.



(e) 20 second time line of processes run/s for Sys_Const and Sys_Inc at Sys_Inc workload=5,000 under a LH-SR allocation. Sys_Inc always has more processes run than Sys_Const.



(f) Processes run/s and actual CPU utilization as seen at the hypervisor level. There is a strong correlation between processes run/s and CPU utilization (Pearson correlation coefficient=0.56).

Fig. 8: Average processes run per second for the database VMs for the LH-SR and HH-SR allocations respectively. When the soft resource allocations are different, the higher concurrency system, Sys_Inc, has more processes run per second at high workloads than the lower concurrency system, Sys_Const. Figures 8(c), 8(d), 8(e) show as the workload of Sys_Inc increases, the processes run per second increases and surpasses Sys_Const's processes run per second. Figure 8(f) shows a strong correlation between processes run per second and actual CPU utilization of the VM which suggests that when processes run per second is high, CPU utilization is also high.

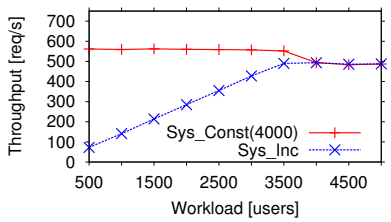
5,000 respectively. When the workload of Sys_Inc is lower than Sys_Const, there are more processes run for Sys_Const; however, when the workloads for the two systems are the same as shown in Figure 8(d), more processes are being run for Sys_Inc than for Sys_Const. When the workload of Sys_Inc is higher than Sys_Const's workload, there are always more active processes for Sys_Inc than for Sys_Const. Figure 8(f) shows a time line graph of the processes run per second and the actual CPU utilization of Sys_Inc's MySQL VM. It shows a strong correlation (Pearson correlation coefficient of 0.56 for the entire experiment run of 300 seconds) between active processes and actual CPU utilization which confirms the behavior of the KVM CPU scheduler.

Once we know that the CPU "stealing" is caused by a discrepancy in the number of active processes being scheduled between the two systems, one straightforward solution is to limit the number of threads that can be run for each VM process under the KVM hypervisor. We can use the setrlimit function[1] to limit the number of threads for a process. When the two systems have the same soft resource allocation, the maximum average number of active processes is 2.5 for both systems as shown in Figure 8(b). Thus, we set the upper bound of the number of threads that can be run per process to 4; after the change, the two systems show similar throughput and response time behaviors as seen when the two systems have the same soft resource allocations as shown in Figure 9(a) and 9(b). The throughput of Sys_Inc does not surpass the throughput of Sys_Const and the response times of the two systems are similar until the workload of Sys_Inc surpasses the workload of Sys_Const. This performance trend is the same as when the two systems have the same soft resource allocations. Figure 9(c) shows the CPU utilization of the database VMs when the workload of Sys_Inc is 5,000. We see that all four VMs are sharing roughly 50% CPU and no one single VM is getting more CPU unfairly. Thus, setting a limit on the threads per VM in the hypervisor mitigates the CPU "stealing" even when the soft resource allocations for the consolidated n-tier applications are different.

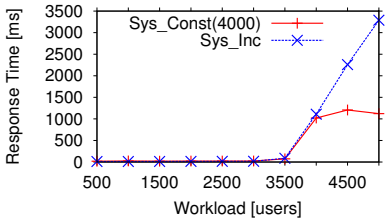
[1]In Linux kernels greater than 2.6.38, it is possible to do the same thing as setrlimit using the prlimit command. Using the setrlimit and prlimit commands is different than ulimit, since ulimit affects the system globally whereas setrlimit and prlimit affect specific processes.

VII. Conclusion

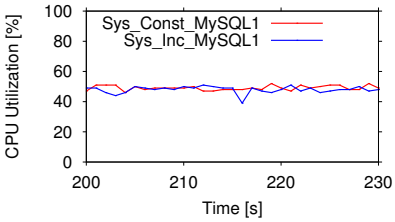
Consolidating n-tier applications in a cloud environment requires a careful tuning of both hardware and software resources. We conducted an empirical study of two consolidated n-tier applications and how changing the soft resource allocations of each application impacted each system's performance in terms of throughput and response time. We found that when the two systems had the same soft resource allocation, a lower concurrency for both systems maximized performance; however, when the two systems had different allocations of soft resources, the system with more soft resources would



(a) Throughput of LH-SR allocation after limiting the number of threads that can spawn for each VM process at the hypervisor level to 4. The throughput has returned to the case where the two systems had the same exact SR allocation and shared each core 50/50.



(b) Response time of LH-SR allocation after limiting the number of threads that can spawn for each VM process at the hypervisor level. The response time is also similar to the case where both systems have the same exact SR allocation.



(c) CPU utilization at the hypervisor level of LH configuration for Sys_Inc workload=5,000. The VMs are now again sharing on average 50% of the CPU core once we limit the number of threads that spawn for the VMs at the hypervisor level.

Fig. 9: Figures 9(a) and 9(b) show the throughput and response times when we limit the number of threads that can spawn for the VM process at the hypervisor level. Once we limit the upper bound for the number of threads that can be created, the throughput and response time behaves in the scenario where the two systems have the same exact soft resource allocation.

achieve better performance due to CPU "stealing". In future work, we will explore the impact of soft resource allocation on other n-tier applications and hypervisors. More generally, our results show that software resource allocation should be considered as a first class citizen when deploying and tuning n-tier application performance in cloud.

ACKNOWLEDGMENT

This research has been partially funded by National Science Foundation by CNS/SAVI (1250260, 1402266), IUCRC/FRP (1127904) , CISE/CNS (1138666), NetSE (0905493) programs, and gifts, grants, or contracts from Singapore Government, Fujitsu Labs, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

REFERENCES

[1] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, April 2007, pp. 200–209.

[2] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding performance interference of i/o workload in virtualized cloud environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 51–58.

[3] Y. Kanemasa, Q. Wang, J. Li, M. Matsubara, and C. Pu, "Revisiting performance interference among consolidated n-tier applications: Sharing is better than isolation," in *Services Computing (SCC), 2013 IEEE International Conference on*, June 2013, pp. 136–143.

[4] Q. Wang, S. Malkowski, Y. Kanemasa, D. Jayasinghe, P. Xiong, C. Pu, M. Kawaba, and L. Harada, "The impact of soft resource allocation on n-tier application scalability," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 1034–1045.

[5] "Rubbos: Rice university bulletin board system," http://jmob.ow2.org/rubbos.html.

[6] M. Sopitkamol and D. A. Menascé, "A method for evaluating the impact of software configuration parameters on e-commerce sites," in *Proceedings of the 5th International Workshop on Software and Performance*, ser. WOSP '05. New York, NY, USA: ACM, 2005, pp. 53–64.

[7] J. Li, Q. Wang, D. Jayasinghe, S. Malkowski, P. Xiong, C. Pu, Y. Kanemasa, and M. Kawaba, "Profit-based experimental analysis of iaas cloud performance: Impact of software resource allocation," in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, June 2012, pp. 344–351.

[8] A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan, "Characterizing tenant behavior for placement and crisis mitigation in multitenant dbmss," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 517–528.

[9] V. Ishakian and A. Bestavros, "Morphosys: Efficient colocation of qos-constrained workloads in the cloud," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 90–97.

[10] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "A cost-sensitive adaptation engine for server consolidation of multitier applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '09. New York, NY, USA: Springer-Verlag New York, Inc., 2009, pp. 9:1–9:20.

[11] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation." in *Int. CMG Conference*. Computer Measurement Group, 2007, pp. 399–406.

[12] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via vm multiplexing," in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 11–20.

[13] T. C. Ferreto, M. A. S. Netto, R. N. Calheiros, and C. A. F. De Rose, "Server consolidation with migration control for virtualized data centers," *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1027–1034, Oct. 2011.

[14] Z. Gong and X. Gu, "Pac: Pattern-driven application consolidation for efficient cloud computing," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, Aug 2010, pp. 24–33.

[15] Q. Wang, Y. Kanemasa, J. Li, C. A. Lai, M. Matsubara, , and C. Pu, "Impact of dvfs on n-tier application performance," in *Proceedings of the 2013 ACM Conference on Timely Results in Operating Systems*, ser. TRIOS '13, 2013.

[16] J. Park, Q. Wang, D. Jayasinghe, J. Li, Y. Kanemasa, M. Matsubara, D. Yokoyama, M. Kitsuregawa, and C. Pu, "Variations in performance measurements of multi-core processors: A study of n-tier applications," in *Services Computing (SCC), 2013 IEEE International Conference on*, June 2013, pp. 336–343.

[17] S. Malkowski, Y. Kanemasa, H. Chen, M. Yamamoto, Q. Wang, D. Jayasinghe, C. Pu, and M. Kawaba, "Challenges and opportunities in consolidation at high resource utilization: Non-monotonic response time variations in n-tier applications," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 162–169.