Maximizing Server Utilization while Meeting Critical SLAs via Weight-Based Collocation Management

Sergey Blagodurov[1], Daniel Gmach[2], Martin Arlitt[2], Yuan Chen[2], Chris Hyser[2], Alexandra Fedorova[1]

[1] School of Computing Science
Simon Fraser University
Vancouver, BC, Canada
{firstname_lastname}@sfu.ca

[2] Sustainable Ecosystems Research Group
Hewlett-Packard Laboratories
Palo Alto, CA, USA
{firstname.lastname}@hp.com

Abstract—**Servers in most data centers are often underutilized due to concerns about SLA violations that may result from resource contention as server utilization increases. This low utilization means that neither the capital investment in the servers nor the power consumed is being used as effectively as it could be. In this paper, we present a novel method for managing the collocation of critical (e.g., user interactive) and non-critical (e.g., batch) workloads on virtualized multicore servers. Unlike previous cap-based solutions, our approach improves server utilization while meeting the SLAs of critical workloads by prioritizing resource access using Linux cgroups weights. Extensive experimental results suggest that the proposed work conserving collocation method is able to utilize a server to nearly 100% while keeping the performance loss of critical workloads within the specified limits.**

Keywords: collocation; SLA; virtualization; server efficiency

I. INTRODUCTION

The rising penetration of digital services leads to a rapidly growing demand in data center computing and increases the already substantial global power consumption of data centers. Despite the significant demand for computational resources, servers in data centers tend to be under-utilized. Low utilization means that power is not used as effectively as it could be, as servers use a disproportionate amount of power when idle, relative to when they are busy. For example, energy efficiency (work completed per unit of energy) of commodity server systems at 30% utilization can be less than half that at 100% [1][2]. Low utilization also contributes to over-provisioning of IT equipment and increased capital expenditures (CapEx).

The advent of virtualization enabled consolidation of several applications onto a server, increasing its utilization. However, even in virtualized resource pools, utilization is typically below 40% [17][18] due to: (1) Most virtualized resource pools are fairly static and don't implement live migration of virtual machines (VMs). (2) Most data center workloads have very bursty demands thus leading to conservative consolidation decisions. (3) The performance of user-interactive applications drops significantly when they must contend for busy resources. The poor performance of these 'critical' applications at high utilization levels is the main challenge in increasing the utilization of servers in data centers.

In this work, we address the challenge of increasing server utilization while maintaining Service Level Agreements (SLAs) by pairing critical workloads such as transactional or interactive workloads with non-critical workloads such as batch processing jobs or High Performance Computing (HPC) applications. Typically, a critical workload has strict performance requirements specified in an SLA, whereas non-critical workloads are more delay tolerant. Most workloads can be categorized into one of these two groups [1][3].

In this work, we assume a virtualized data center that uses a *work conserving approach* (i.e., physical resources are shared among hosted VMs). Much previous work in virtualized data center management uses a *non-work conserving approach* commonly referred to as resource capping where each workload has restricted access to the resources it can use per interval. To guarantee workload isolation, the sum of the caps per each resource must not exceed the available resource capacity. Unfortunately, enforcing caps typically leads to low average utilizations. The *work conserving approach* however, typically leads to higher utilization but no resource access guarantees can be made. In practice, an expert chooses the consolidation factor, i.e., the utilization level up to which workloads will be consolidated.

Our solution is to consolidate both batch and interactive workloads onto each server, enabling a very high utilization level (80% and above). At higher utilizations, the performance of the hosted workloads is likely to degrade due to contention for shared server resources. We prevent this by providing prioritized access to physical resources using Linux Control Groups (cgroups) *cpu.shares* [38]. Since the term *shares* is often used in cap-based work to denote a portion of CPU allocated to a particular job, we refer to Linux CPU shares as CPU weights in this paper, to avoid confusion. We ensure that critical workloads have preferred access to physical resources such that they exhibit similar performance when consolidated with non-critical workloads as compared to when they are not. We demonstrate that the performance for low priority, non-critical workloads, which are typically less response-time sensitive, remains acceptable as long as server resources are not excessively over-provisioned.

In this work, we focus on increasing the CPU utilization while considering other physical resource constraints. We note that our approach can also be applied in the presence of network and I/O utilization bottlenecks. Our work provides several unique contributions:

1) We demonstrate that static cgroups weights can be used to increase overall server utilization while maintaining the performance of critical workloads. We quantify the benefits

and show the impact to critical and non-critical workloads.

2) We evaluate the correlation between the CPU access weights and the workload CPU consumption. We further demonstrate that intelligent assignment of critical workload virtual CPUs to physical CPUs can completely mitigate the impact of collocation even at very high utilization levels.

3) We show that dynamic weight management can preserve SLAs when multiple bursty critical workloads share resources. We present a model that dynamically assigns weights based on performance and past resource consumption. We show that this approach can be used to provide fairness or performance isolation between workloads.

The remainder of our work is organized as follows. Section II presents related work. An overview of the system under study is provided in Section III. Section IV describes our provisioning method for maximizing server utilization using static resource access weights. Section V explores dynamic CPU access management of critical workloads, followed by a summary of our work in Section VI.

II. Related Work

Most prior work in workload management uses some form of capping to distribute resources among collocated workloads in non-work conserving mode [3–9][11][13–15][31]. Examples are limiting CPU resource utilization in Xen with CPU caps or limiting network bandwidth through Linux Control Groups. With such approaches, non-critical workloads only have access to limited amounts of server resources. If the controller does not perform runtime adjustments of the caps, the utilization of the non-critical workloads will stay low even when resources are available, resulting in lower overall server utilization.

Other work improves server utilization by using weights to control resource assignment in work conserving mode. The work by Diao et al. prioritizes and dispatches incoming requests to service agents based on the SLA attainment target in a simulation testbed [39]. Others use existing techniques such as CPU shares, Xen CPU weights, or Linux real-time priorities to collocate interactive jobs [10][16] and mix them with batch jobs [26][36]. By controlling resource access priorities of collocated workloads, a controller is able to adjust to workload demand changes allowing applications to use resources when available. In this work, we introduce a novel approach that fully loads the server with critical and non-critical loads. None of the previous work investigated whether a work-conserving approach can be used to significantly increase resource utilization up to full server capacity while maintaining acceptable performance. We have developed a working prototype and we show that it is feasible to achieve high server utilization with a detailed experimental study using real workloads. We leave the evaluation of financial and power saving benefits of full server utilization for future work.

Our solution works very well with traditional sizing tools, e.g., Capacity Manager [33][35], which determine the optimal amount of critical workload and ensure that not too many critical jobs are placed on a system. Further, sizing tools can ensure that batch jobs get enough resources to avoid starvation. We then use reactive, weight-based workload management to maintain performance of the critical applications. Our weight-based management solution is complimentary to more sophisticated predictive controllers that consolidate workloads based on their resource profiles.

Cap-based techniques that enable existing middleware to manage mixed batch and transactional workloads have been previously proposed [3]. A workload profiler monitors the nodes' resource utilization and estimates an average CPU requirement for each application request. The system then predicts the performance of the transactional application for a given allocation of CPU resources. The authors use simulation to demonstrate the benefits of their approach.

Bubble-up [4] tries to collocate workloads onto the same server that, according to prediction, will not conflict. A profiling run for each application is required for making the prediction. During the run, the application is collocated with a special "bubble" that generates a profiling report. While the proposed approach works well for the examined Google workload, application profiling may be challenging in general, because data center workloads may be unknown in advance.

In order to address QoS requirements in Q-Clouds [5], VMs are first profiled in a solo run to determine the amount of resources needed to attain the desired QoS. The Q-Clouds scheduler then leaves a prescribed amount of unused resources on each server. The Q-Clouds implementation also dynamically adjusts VCPU caps of the VMs through the Hyper-V hypervisor.

Several hybrid provisioning approaches [6–8][33] combine prediction and reaction while performing cap-based SLA management in a data center. They use historical trace analysis to predict future workload demands and to determine the amount of workload given to any particular server. In addition, they implement a reactive controller that handles unforeseen events.

The Active CoordinaTion (ACT) approach described in [9] relies on the notion of Class-of-Service, a multi-dimensional vector that expresses the resource expectations of Xen VMs. Based on these vectors it caps VMs in their resource usage. ACT uses historic VM behavior to infer future trends.

Sandpiper [10] relocates VMs from overloaded to under-utilized nodes using a black-box approach that is OS and application agnostic and a gray-box approach that exploits OS and application-level statistics. Explicit SLA violations must be detected on a per-virtual server basis in the gray-box approach—a hotspot is flagged if the memory utilization, response time or request rate exceed an SLA threshold. Before resolving the hotspot via migrations, Sandpiper estimates the additional resource needs for overloaded VMs using a high percentile of the CPU and network bandwidth distribution as a proxy for the peak demand.

The creators of GreenSlot [31] present a parallel batch job scheduler for data centers powered by PV energy. They predict the amount of available energy in the near future and schedule workloads to maximize green energy consumption while meeting the jobs' deadlines. If grid energy must be used, the scheduler selects times when it is inexpensive.

Table I. Critical and Non-critical Workloads used in this Study.

Class	Application	Description
Critical	RUBiS	RUBiS is modeled after eBay.com and implements the core functionality of an auction site: selling, browsing and bidding. It is widely used as a benchmark in research [19][20][21]. We used a 3-tier RUBiS setup in our experiments where client requests are first served by an Apache Web server. It answers static HTML pages directly and uses a JBoss application server for dynamic requests. The data for the dynamic pages is stored in a MySQL database. The load on the auction Web site is varied by changing the number of concurrent clients.
	Wiki	Wiki is a realistic Web hosting benchmark based on WikiBench [30] and MediaWiki, which is the application used to host wikipedia.org. It uses real Wikipedia database dumps and generates traffic using publicly available traces. Our Wiki setup consists of a workload generator and three tiers: an Apache Web and application server (PHP-based), a MySQL database server and a Memcached server [25].
Non-critical	Swaptions	Swaptions is a financial analysis benchmark suite from Intel that mimics an RMS (recognition, mining and synthesis) workload that uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of Swaptions [23]. Swaptions is a multi-threaded application.
	Facesim	Facesim is animation software that takes a model of a human face and a time sequence of muscle activations and computes a visually realistic animation of the modeled face [23]. Facesim is a multi-threaded application.
	Fire Dynamics Simulator	Fire Dynamics Simulator (FDS) is a fire simulator that computes a computational fluid dynamics model of fire-driven fluid flow [24]. FDS is a single-threaded application.
	LU, BT, CG	LU (Lower-Upper Gauss-Seidel solver), BT (Block Tri-diagonal solver) and CG (Conjugate Gradient, irregular memory access and communication) are popular scientific programs written in MPI (Message Passing Interface) programming model [27].

Table II. System Parameters

Parameter	Range of Values in our Experiments
Server utilization	Ranges from 1% in case of the solo RUBiS run with 45 requests per second up to 100% in case of the collocated runs of one or two critical applications with three batch jobs.
Measurement granularity	We measure performance metrics of critical and non-critical applications every 5 seconds.
Cgroups cpu.shares (CPU weights)	We vary the weight values from 2 (the minimum possible value) up to 262,144 (the maximum possible value) for each process.
Linux kernel versions	3.4, 3.0

By colocating with other tenants of an Infrastructure as a Service (IaaS) offering, IaaS users could reap significant cost savings by judiciously sharing their use of the fixed-size instances offered by IaaS providers [11]. Each server instance is characterized by a number of resources (CPU, network, memory, and disk space) which constitute dimensions of the instance capacity vector. Workloads associated with an instance are similarly characterized by a multi-dimensional utilization vector. User needs are spelled out as SLAs defined over the various resources of the instance (e.g., CPU, memory, local storage, network bandwidth). Based on this input, a desirable VM configuration is calculated, reflecting what the service deems best for each customer in the system. The framework then carries out the necessary migrations of VMs.

III. System overview

A. *Workload Description*

Table I describes the workloads that we use for our experiments. We use RUBiS and Wiki as representatives for critical workloads and combinations of financial, animation, simulation and scientific applications as non-critical workloads [22–24][27].

Both critical applications are Web applications. Wiki represents a fairly large and computational intensive application that triggers multiple queries to the database or Memcached server to create its dynamic Web pages. Wiki response times in our setup are typically on the order of 500 milliseconds (ms) and above. In contrast, RUBiS is a much faster application with response times on the order of tens of milliseconds.

We divide our non-critical workloads into two groups. In Group A we use a set of Swaptions, Facesim, and FDS. These are popular applications representing three typical types of the data center batch loads: finance, animation and simulation. Swaptions and Facesim are multi-threaded, mostly CPU intensive, with Facesim and FDS being partially I/O bound. They have no network communication. Group B is comprised of LU, BT and CG, which are CPU bound, network intensive MPI jobs. They represent a typical HPC workload. MPI applications are becoming increasingly important in the data center as more and more HPC workloads are being moved "into the cloud" (i.e., hosted in a remote, shared, virtualized data center).

Next, we describe the performance metrics that we use to measure the performance level of the workloads. For transactional workloads, we measure the average, 95th and 99th percentile response time and request rate every 5 seconds. We obtain the response times for each request by using information from the client (i.e., the workload generator); however, in a real environment, response times from the Web server log files can be used. We created a monitor that parses the workload generator log files and calculates the performance metrics at runtime.

The number of CPU cycles consumed is used to estimate the progress of batch workloads. Empirical studies reveal that these workloads consume a certain amount of CPU time (with small variation) to complete. Hence, we use CPU consumption in each measurement interval as a performance metric for the

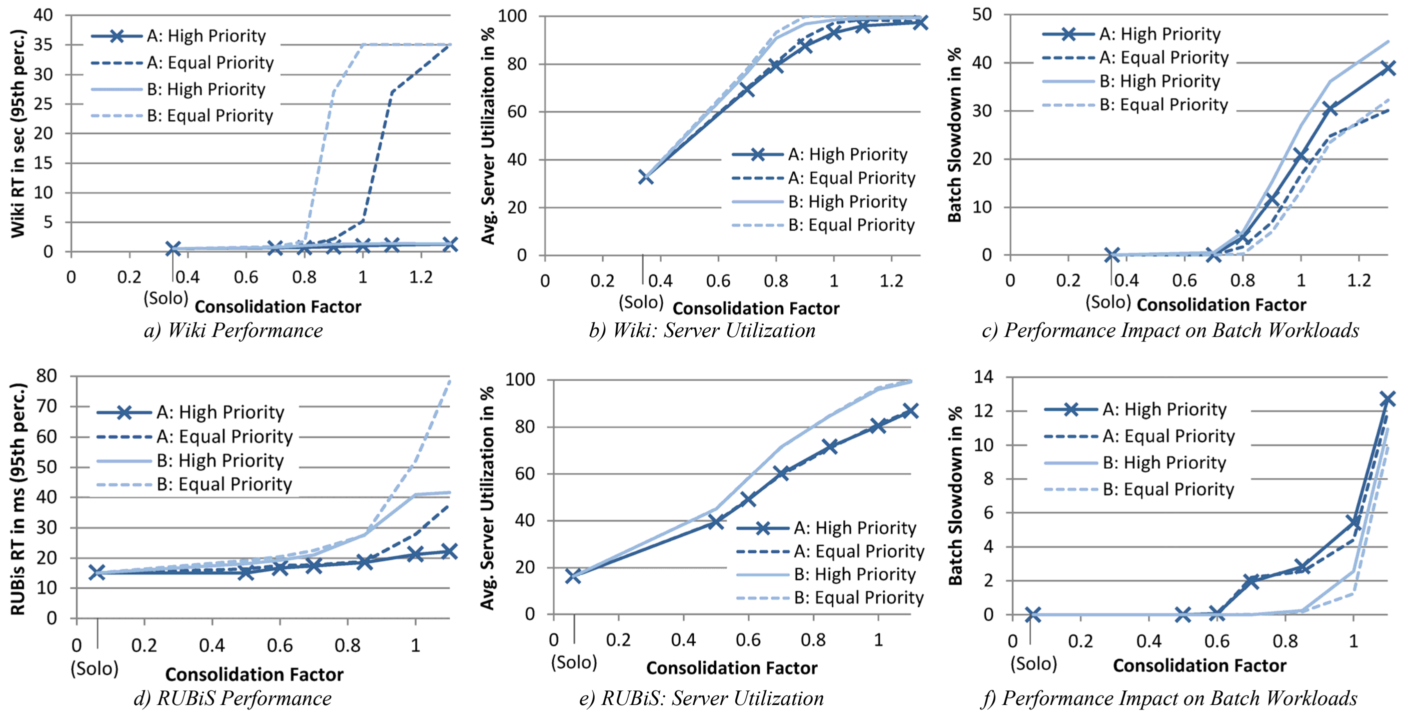


Figure 1.Workload Collocation using Static Prioritization for Scenario A (Swaptions, Facesim, FDS) and Scenario B (LU, BT,CG)

batch workloads under study. We compare these values with the "solo performance" of the batch workload (i.e., performance obtained when the batch workload has exclusive use of the server) to determine the degree of slowdown resulting from collocation. For example, if the CPU utilization of a batch workload is decreased to half the utilization of the solo run then the slowdown would be 50%.

B. Experimental Testbed

Our testbed consists of five HP ProLiant BL465c G7 servers. Each is equipped with two 12-core AMD Opteron 6176 (2.3 GHz) CPUs. Each server has 5 MB shared cache and 64 GB of main memory distributed across 4 NUMA memory nodes. All servers are connected via a 10 Gb/s Ethernet network and have access to 4 TB of NFS storage.

We consider an environment where applications are hosted within a common pool of virtualized servers. Each application or application tier runs in a virtual machine (VM). The resources of a physical server, including CPU, memory, disk and network I/O bandwidth, are shared by the hosted VMs. In our prototype implementation, we use KVM as the virtualization platform [28] running on Ubuntu 11.10. We further installed the latest stable Linux kernel version 3.4.

We prioritize the access to CPU cycles for the collocated workloads by adjusting the parameter *cpu.shares*, which is a configuration parameter available through Linux Control Groups (*cgroups*). It specifies how VMs are given processor time by the scheduler. In our setup, we are able to vary CPU shares values from 2 up to 262,144 for each process. These values are relative to each other. A value twice as high for a process compared to another denotes that it has access to twice as many CPU cycles as the other process. In the literature, this is often referred to as work conserving mode [32].

Linux Control Groups also allow prioritizing access to resources other than CPU. Currently, priority-based management of I/O and network access is supported via *blkio* and *net_prio* modules, respectively. Although we did not observe significant contention for disk and network in our experiments, we note that the method introduced in this paper can be directly extended to non-CPU resources by using these Control Groups capabilities.

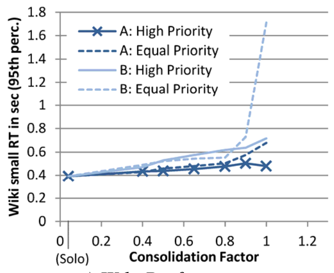
Table II. System Parameters summarize the system parameters that are considered and the values used in our experiments.

IV. DRIVING SERVER UTILIZATION UP

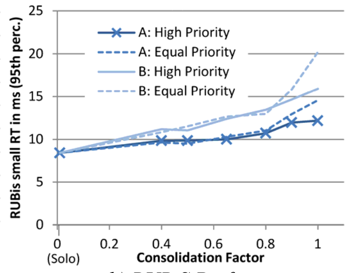
This section explores how well we can prioritize the access to CPU between collocated workloads using static CPU weights. We evaluate in detail the performance impact on critical and non-critical workloads.

A. Workload Collocation using Static Weights

For the experiments shown in Figure 1, we vary the load on the server by changing the number and size of batch workloads consolidated with the critical workload on the server. To change the size of batch workloads, we vary the number of virtual CPUs assigned to each batch VM. A consolidation factor of one indicates that the sum of the demands of all hosted workloads equals the machines capacity, i.e., the workloads hosted on the machine are expected to consume all available CPU resources. A consolidation factor above one indicates that demand for CPU resources is higher than the server capacity and thus some workloads will not get their CPU demand fully satisfied. As a baseline we run the critical workload in isolation. We then evaluate its performance when collocating with non-critical workloads.



a) Wiki Performance



b) RUBiS Performance

Figure 2. Impact of Critical Workload Size

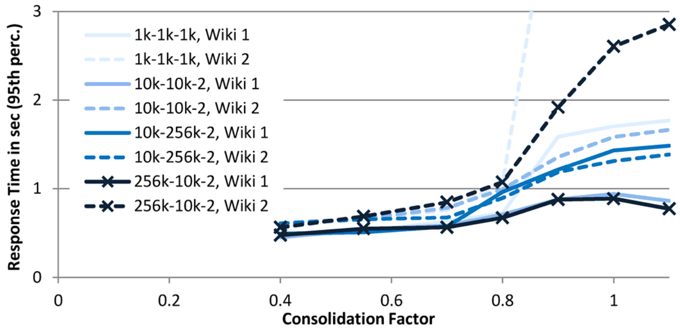


Figure 3. Collocating two Critical WLs with Non-critical WLs. (Scenario B)

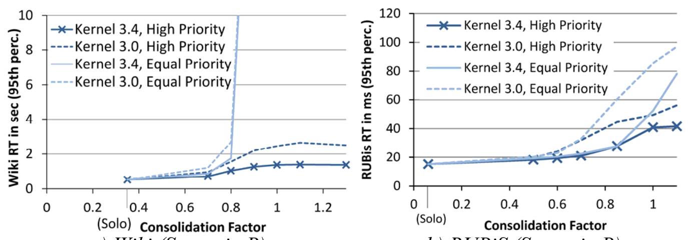
Further, we consolidate the critical workload with two different sets of non-critical workloads. In Scenario A, we use Swaptions, Facesim, and FDS, whereas in Scenario B we use LU, BT, and CG as the non-critical workloads. Figure 1 compares two static weight approaches. The first approach *Equal Priority* follows the default settings in cgroups and assigns a CPU weight of 1,024 to each process. The *High Priority* approach provides preferred CPU access to critical workloads. Since the critical applications have several tiers, we set the CPU weight value for each tier individually. We use a CPU weight of 10,000 for critical workloads and a weight of 2 for non-critical workloads, i.e., a critical workload is allowed to use 5,000 times more CPU cycles per interval than a non-critical workload. This mostly ensures that non-critical VMs will only be given access to CPU cycles that are not requested by critical VMs.

Figures 1a)–c) present the results of collocating Wiki with non-critical workloads. We performed each experiment multiple times and observed little variation in the results. Wiki is an open loop benchmark and the Wiki client (WikiBench) is configured to submit 40 requests per second. Figure 1a) shows the 95th percentile response time of Wiki in seconds with respect to the consolidation factor. We use a standard MediaWiki setup that has a 95th percentile response time of 0.5s when running solo on the server, resulting in an overall system utilization of about 35% as shown in Figure 1b). In accordance with Human Computer Interaction (HCI) research, we consider response times below 2s as acceptable application responsiveness to humans [34] (lower values are desirable, while values over 2s are deemed unacceptable). When consolidating Wiki with non-critical workloads, the results show that its performance remains acceptable until a consolidation factor of 0.8. For example, in Scenario A the 95th percentile response time increases to 1.2s when using equal weights compared to 0.75s when prioritizing Wiki. However, as soon as the server starts getting overloaded, i.e., the server utilization gets above 90% in Scenario A or 80% in Scenario B, the performance of Wiki suffers greatly with equal weights. We note that the response time does not exceed 35s because requests time out in the WikiBench client. Giving Wiki preferred access to CPU improves its performance vastly. Even in the highly overloaded case when consolidating workloads of 1.3 times the server's capacity, the 95th percentile response time is still below 1.4s in both scenarios. While we do not recommend operating servers at this load level, we note that unexpected load increases might push a server into such overload situations. Providing preferred access to critical workloads helps mitigate their performance degradation until remedial actions can be taken. Figure 1c) shows the performance loss for the non-critical applications. As soon as the server utilization approaches 100%, the non-critical workloads suffer from access to fewer resources. The figure shows that the slowdown of non-critical workloads increases by up to 12% when providing preferred CPU access to critical workloads. For example, in the *High Priority* case with a consolidation factor of 1.3 in Scenario B the batch workloads experience a slowdown of 44% compared to 32% with equal weights. We believe that the additional performance degradation of the batch jobs through prioritizing critical workloads is tolerable as various data suggests that the batch deadline estimates are usually padded by 20 to 40% across a wide spectrum of systems [29][31].

Figures 1d)–f) shows the corresponding results using RUBiS as a critical workload. The RUBiS client is configured to trigger an average of 450 requests per second (req/s). RUBiS has a much smaller footprint than Wiki, requiring only 6% of the server's CPU resources. In the solo run, RUBiS achieves a 95th percentile response time of 15 ms. When prioritizing RUBiS, its response time at a consolidation factor of 1.1 is still below 34 ms for Scenario A and 50 ms for Scenario B. The difference in impact between the two scenarios is due to workload A being partially I/O bound. As a result, it generates less contention than B for computational resources of the server. We note that the RUBiS response times are still short enough that they fall into the "crisp" response time category defined by the HCI community [34]. This means that a human user of the RUBiS application would be unlikely to notice any difference between the response times in the solo case and the prioritized collocation case, even at a collocation factor of 1.1.

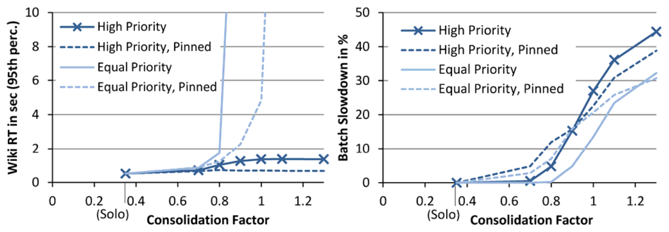
B. Impact of Critical Workload Size on Performance

The likelihood that workloads with equal CPU demand per request get all their demands satisfied depends on their size. For example, if Workload 1 services twice as many requests as Workload 2 we would need to give Workload 1 twice as large a weight as Workload 2, such that both have an equal probability to satisfy their demands. Figure 2 shows the impact of the critical workload size on its performance. For both Wiki and RUBiS we reduce the request rate to 10% of its original rate. The figures show that the benefits of prioritization depend on the size of the critical workloads. Although both applications improve their performance when having preferred access to CPU, the differences to the default equal priority settings is lower. This indicates that adjusting CPU weights is more important for larger critical workloads.



a) Wiki (Scenario B) *b) RUBiS (Scenario B)*

Figure 4. Impact of Kernel Version



a) Wiki Performance *b) Non-Critical WL Performance*

Figure 5. Pinning VCPUs of Critical Workloads (Scenario B)

C. Providing Different Weights to Critical Workloads

This section evaluates the impact of different CPU weights for critical workloads. We consolidate two Wiki applications. Wiki 1 has a request rate of 4 req/s resulting in 4% server utilization, and Wiki 2 has 40 req/s resulting in 35% utilization. Further, we consolidate them with non-critical workloads from Scenario B and vary the consolidation factor from 0.4 to 1.1 by changing the size of the non-critical workloads. Figure 3 compares four different configurations:

1) *1k-1k-1k:* Each workload has a default weight setting of 1,024 (1k). This represents the *Equal Priority* approach from the previous sections.

2) *10k-10k-2:* Both critical applications Wiki 1 and Wiki 2 have a CPU weight of 10k and the non-critical workloads have a weight of 2. These are the same weight settings as in the *High Priority* approach.

3) *10k-256k-2:* The smaller Wiki 1 has a CPU weight of 10k and the larger Wiki 2 has a weight of 256k. Non-critical workloads have a CPU weight of 2.

4) *256k-10k-2:* The smaller Wiki 1 has a CPU weight of 256k and the larger Wiki 2 has a weight of 10k. Non-critical workloads have a CPU weight of 2.

The first Configuration *1k-1k-1k* provides the worst performance of critical workloads. As indicated in Section IV.B, the critical workload size has an effect on its performance. Given equal weights of 10k in Configuration 2) the smaller Wiki 1 consistently achieves better response times. The differences are even larger in Configuration 4) where the smaller application Wiki 1 has the highest weights. Interestingly, its performance improvement is marginal whereas the Wiki 2 suffers greatly. Providing preferred access to the larger critical application Wiki 2 allows balancing performance between the critical workloads. In Configuration 3) both Wiki applications exhibit comparable performance.

D. Impact of Kernel Version on Performance

The feature for fine-grained CPU resource control via Linux control groups was added to the kernel in version 2.6.24 and since then many performance improvements to the kernel scheduler have been made. To illustrate the progress we compare the results of kernel 3.4 from Figure 1a) and 1d) in Scenario B with corresponding results with the default Ubuntu 11.10 kernel version 3.0. The results are shown in Figure 4. We note that the performance of critical workloads improved in all scenarios, especially when providing preferred access to CPU. For example, in Figure 4a) the response time of Wiki with high priority at very high system load is twice as long with the 3.0 kernel compared to the 3.4 kernel. The performance differences for RUBiS between the two kernel versions are smaller than for Wiki but still significant, as shown in Figure 4b). While the critical application benefitted from a newer kernel version, the performance of the batch workloads remained unchanged. This indicates that switching costs between Linux processes and the allocation of virtual machine CPUs to physical CPU cores has improved significantly with newer kernel versions [12][37]. Indeed, we have measured a 39% reduction in cross-core VCPU migrations in the kernel 3.4 relative to 3.0. The average VCPU waiting time in a physical core run queue has also decreased by 17%.

E. Pinning VCPUs of Critical Workloads

In this section, we investigate the reasons for the performance decrease of critical workloads with high weights when increasing workload collocation. In particular, we noticed that the virtual CPUs (VCPUs) belonging to the VMs of critical workloads often were allocated on a subset of the available physical cores, which resulted in the critical VCPUs competing with each other for CPU cycles. To determine whether this affected performance, we pinned each critical VCPU to a single physical CPU core. In our setup, the Wiki application has fewer VCPUs assigned to it than the number of physical CPUs on the server. The VCPUs of non-critical workloads remained unbound such that the kernel scheduler is able to map them freely. Figure 5a) shows that with preferred access to CPU and VCPUs pinned, Wiki's performance is almost independent of the physical server utilization; compared to the solo run (0.52 seconds) its 95th percentile response time increases to 0.68 seconds at a consolidation factor of 0.7. Even at a very high consolidation factor of 1.3 its response time remains at 0.68 seconds.

Figure 5b) indicates that the performance impact on non-critical workloads is small when pinning critical workload VCPUs. While batch workloads are slowed down more at consolidation factors below 0.9, they actually improve their performance in the *High Priority* scenario for higher consolidation factors when pinning critical VCPUs.

Figures 5a) and 5b) show that we can make resource sharing more efficient by improving the allocation of VCPUs to physical CPUs when prioritizing workloads. However, in the case where the total number of critical VCPUs exceeds the available number of physical CPU cores, the optimal allocation of critical VCPUs to physical CPU cores is non-trivial. We believe that further performance improvements can be made by making the process-to-CPU mapping algorithm of the Linux scheduler aware of cgroups CPU access weights. We will evaluate this further in our future work.

F. *Summary of Collocation with Static Weights*

The previous sections showed that collocating critical and non-critical workloads achieved server utilizations of 80% and above without significant performance decreases for the critical workloads. Although the 95th percentile response times of critical workloads increased between 41% and 98% when comparing a solo run and a collocated run with up to 80% server utilization, it is important to note that based on HCI research [34] we anticipate minimal consequences for end users. In particular, in the solo experiments the physical resources are highly over provisioned. This results in much lower server utilization. While our approach increases the response times to improve server utilization, we maintain the response times in similar HCI "categories" [34] (e.g., "crisp" for the RUBiS application), such that typical users will not notice the difference in performance.

The experimental results show that when collocating workloads, providing critical workloads preferred access to physical resources improves their performance significantly, while having only a minor effect on non-critical workloads. Further, providing prioritized access to resources helps mitigate performance losses during sudden overload situations. Finally, providing prioritized access to CPU resources using static CPU weights in cgroups is available in most Linux distributions and does not incur additional overhead.

V. Dynamic Prioritization of Critical SLAs

Section IV showed the benefits of providing prioritized CPU access to workloads using static CPU weights. In this section, we evaluate the impact on performance when weights are controlled dynamically. We consider two scenarios: in the *Fair* scenario, we adjust CPU weights based on application performance, e.g., the response time. Thus, two applications with similar SLAs should have an equal chance to achieve their SLA goal. In the *Isolation* scenario*,* we consider performance SLAs with a specified maximum load for each application, e.g., a maximum request rate. We then assign CPU weights based on the *Fair* scenario as long as request rates stay below the agreed maximum level. If the request rate of an application exceeds the maximum level then this application receives lower CPU weights in order to reduce the impact on other applications.

A. *Model for Dynamical CPU Weights*

In this section, we assume that each critical workload has an SLA assigned that specifies a desired response time RT^{SLA}. Our approach first determines how much CPU should be allocated to each workload in the next control interval based on its current performance and CPU consumption, similar to a non-work conserving approach. We then translate these CPU allocation values to CPU weights. To demonstrate our approach, we use a simple feedback controller. We note that more sophisticated feed-back or feed-forward controllers as presented in [6–8][33] could be easily integrated.

First, we determine a performance factor *PF* for each critical application i in Equation (1). $RT_i(t)$ is the 95th percentile response time of the application in the current interval t and RT_i^{SLA} is its response time requirement. $PF_i(t)$ ranges from -1 to $+\infty$. A value below 0 indicates that the application meets its response time requirement, whereas a value larger than 0 denotes by how much it misses the requirement.

$$PF_i(t) = \frac{RT_i(t) - RT_i^{SLA}}{RT_i^{SLA}} \qquad (1)$$

Next, we determine how much CPU should be allocated to the workload in the next interval based on its current performance factor and CPU consumption. We note that applications are typically comprised of several tiers. For instance, the Wiki application contains a Web server, a Database and a Memcached tier, each running in its own VM. $C_{i,j}(t)$ is the current CPU consumption of tier j of application i as a fraction of the total server CPU capacity and $C_{i,j}(t+1)$ is the desired CPU allocation for the next interval $t+1$:

$$C_{i,j}(t+1) = C_{i,j}(t) \cdot \left(1 + PF_i(t)\right) \qquad (2)$$

Finally, we convert the desired CPU allocation values into the CPU weights $W_{i,j}(t+1)$. In this work, we use a simple linear model for the translation. The intuition behind the transformation is that in the case when the physical server is fully utilized or even overloaded the CPU weights determine the amount of CPU each workload tier is able to access. Setting CPU weights in relation to the desired CPU allocation values gives each workload tier an identical probability to satisfy all of its demands.

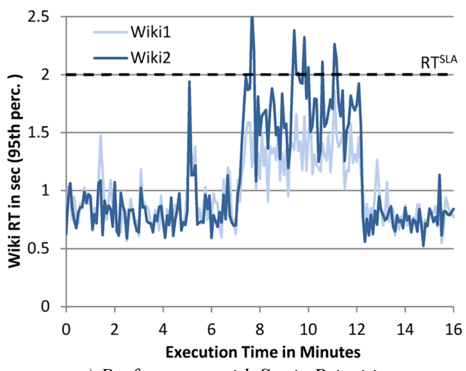
For the linear transformation model shown in Equation (3), we set $W^{max} = 256$k, which is the available maximum CPU weight value on our systems. Further, we set $W^{min} = 1k$, which is the default CPU weight on our systems. The initialization of W^{max} and W^{min} in a different setup must be done based on the available control means and datacenter requirements. A lesser value of W^{max} would imply less room to differentiate between several critical applications, while higher value of W^{min}would additionally mean more isolation for critical jobs from the non-critical influence. $C^{max}(t+1)$ is the maximum desired CPU allocation across all critical application tiers. The CPU weight $W_{i,j}(t+1)$ of application tier i,j for the next interval is defined as:

$$W_{i,j}(t+1) = (W^{max} - W^{min}) \cdot \frac{C_{i,j}(t+1)}{C^{max}(t+1)} + W^{min} \qquad (3)$$

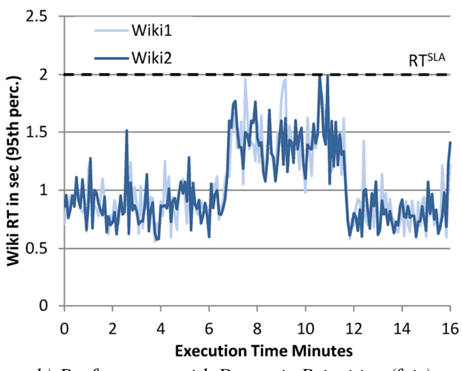
B. *Providing Fairness to Critical Workloads*

This section demonstrates how dynamic CPU weights can provide fairness to the critical workloads, i.e., they are able to achieve a similar level of performance regardless of their load (user requests). For Wiki applications, we specify a 95th percentile response time requirement of 2 seconds. We chose this SLA, because HCI research suggests that response times below 2s are acceptable application responsiveness to humans [34]. We note that even in our solo setup Wiki exhibits fairly long response times. We believe response times could be improved by tweaking the application.

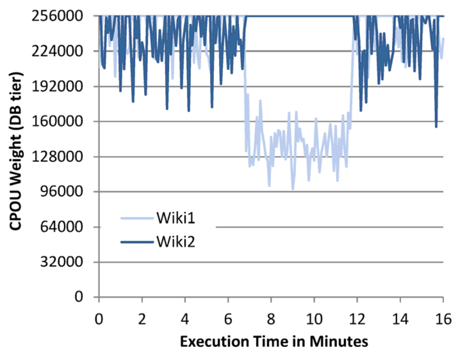
We then consolidate two Wiki applications, which have a default request rate of 20 req/s, together with the batch workloads from Scenario B. Each Wiki utilizes about 20% of the CPU resources of the physical server. The total server utilization is around 80%. After 7 minutes, Wiki 2 experiences a 100% increase in its load for 5 minutes. Figure 6(a) shows the Wiki performances assigning static CPU weights as in the



a) Performance with Static Priorities

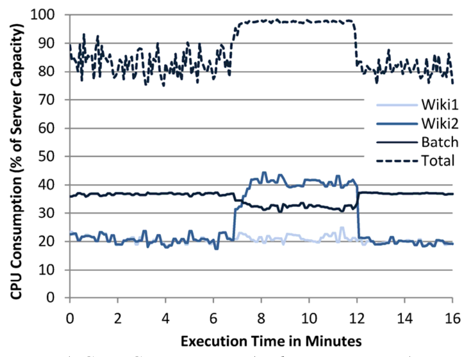


b) Performance with Dynamic Priorities (fair)

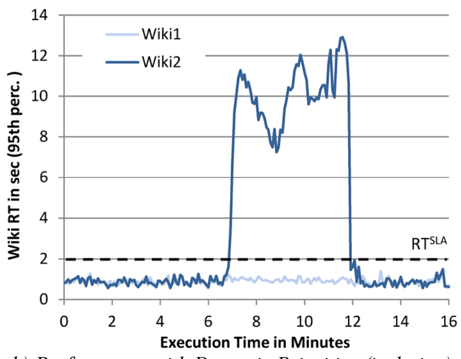


c) CPU Weights of DB Tiers (fair)

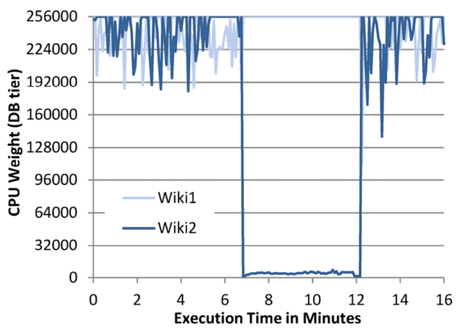
Figure 6. Workload Collocation during Spikes Providing Fairness



a) CPU Consumption (isolation scenario)



b) Performance with Dynamic Priorities (isolation)



c) CPU Weights of DB Tiers (isolation)

Figure 7. Workload Collocation during Spikes Providing Isolation

10k-10k-2 Configuration from Section IV.C. During the spike the performance impact on Wiki 2 is higher than on Wiki 1 resulting in a SLA violation for Wiki 2 18% of the time.

Figure 6(b) shows results for the *Fair* Scenario where CPU weights are assigned dynamically every 5 seconds according to the model of Section V.A. The figure shows that in this scenario the two critical applications, Wiki 1 and Wiki 2, exhibit similar performance over the complete run. During the spike, the CPU weights for Wiki 1 decrease such that Wiki 2 is able to achieve a comparable performance. The CPU weights for Wiki 1 and 2 are shown in Figure 6(c). We note that both applications perform within their response time requirement of 2.0 seconds.

C. Achieving Workload Isolation with Dynamic Prioritization

The big drawback of the work conserving mode is the reduced degree of isolation. For example, workloads that drastically increase their demands can steal CPU resources from other workloads with negative impact. In this section, we consider an SLA where the performance requirement (e.g., response time) is restricted by a certain maximum load (e.g., maximum number of req/s). For the experiment, we set the maximum allowed load to 30 req/s. If a critical application i exceeds the maximum load then we reduce its response time requirement RT_i^{SLA} in Equation (1) to a large value. This results in lower CPU weights and mitigates its impact on other workloads. We note that a migration controller as presented in [6] can be used to migrate off workloads if server load exceeds a maximum threshold for too long. As long as load stays within the maximum load our approach assigns dynamic CPU weights as in Section V.B.

Figure 7 shows the results of the *Isolation* Scenario. When the load of Wiki 2 exceeds 30 req/s, its response time requirement is set to a million seconds, practically infinity. The controller then reduces the CPU weights for Wiki 2 as shown in Figure 7(c). By adjusting the weights, we are able to mitigate the performance impact on Wiki 1 during the spike as shown in Figure 7(b). As expected, the performance of Wiki 2 drops significantly for the time it exceeds the maximum allowed load. Figure 7(a) shows the CPU consumption for each application during the experiment. Despite the lower weights for Wiki 2 during the spike, its CPU consumption increases due to the increased demand.

VI. Conclusion

A long-standing problem associated with data centers is server under-utilization, which increases both CapEx and OpEx costs. In this paper, we present a novel method for improving server CPU utilization through weight-based collocation management of critical (e.g., user interactive) and non-critical (e.g., batch) workloads on virtualized multi-core servers. Our experimental results reveal that the proposed collocation method is able to utilize a server to near full capacity with small critical workload performance loss. In future work, we will consider dynamic pinning of VCPUs to physical CPUs, consider hard deadlines for batch jobs, investigate scenarios where other resources than CPU are the bottleneck, and quantify the energy and cost savings of collocation.

Acknowledgement

We thank Christopher Stewart for valuable comments on feedback controllers in the non-work conserving mode.

REFERENCES

[1] U. Hoelzle and L. A. Barroso. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. 2009.

[2] http://cdn.globalfoundationservices.com/documents/MSFTTop10BusinessPracticesforESDataCentersAug12.pdf .

[3] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé. Enabling resource sharing between transactional and batch workloads using dynamic application placement. Middleware, pages 203–222, 2008.

[4] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In MICRO 2011.

[5] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In EuroSys 2010.

[6] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 islands: Integrated capacity and workload management for the next generation data center. ICAC '08.

[7] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing Data Center SLA Violations and Power Consumption via Hybrid Resource Provisioning. In IGCC 2011.

[8] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. J. Watson, W. Rivera, X. Zhu, and C. D. Hyser. AppRAISE: application-level performance management in virtualized server environments. IEEE Transactions on Network and Service Management, 6, 2009.

[9] M. Kesavan, A. Ranadive, A. Gavrilovska, and K. Schwan. Active coordination (act) - toward effectively managing virtualized multicore clouds. In CLUSTER, pages 23–32, 2008.

[10] T. Wood, P. Shenoy, and Arun. Black-box and Gray-box Strategies for Virtual Machine Migration. pages 229–242.

[11] V. Ishakian, R. Sweha, J. Londono, and A. Bestavros. Colocation as a service: Strategic and operational services for cloud colocation. Network Computing and Applications, IEEE Symposium on, pages 76–83, 2010.

[12] http://www.linux-magazine.com/content/download/65332/508584/file/094-095_kernelnews.pdf

[13] R. Wang, D. Kusic, and N. Kandasamy. A distributed control framework for performance management of virtualized computing environments. In ICAC '10.

[14] D. Minarolli and B. Freisleben, "Utility-based resource allocation for virtual machines in Cloud computing". In ISCC 2011.

[15] E. Loureiro, P. Nixon, and S. Dobson. A Fine-Grained Model for Adaptive On-Demand Provisioning of CPU Shares in Data Centers. In IWSOS '08.

[16] A. Sangpetch, A. Turner, and H. Kim. How to tame your VMs: an automated control system for virtualized services. In LISA'10, 1-16.

[17] http://www.dell.com/downloads/global/solutions/public/white_papers/dell_sql_cons_virt_wp1.pdf

[18] http://www.infoworld.com/print/146901

[19] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. SIGMETRICS 2005.

[20] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. Middleware 2008.

[21] R. Hashemian, D. Krishnamurthy, and M. Arlitt. Web Workload Generation Challenges - An Empirical Investigation. Wiley Software Practice and Experience, 2011.

[22] S. Blagodurov and M. Arlitt. Improving the efficiency of information collection and analysis in widely-used it applications. ICPE 2011.

[23] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. Technical report, Princeton University, 2008.

[24] R. Curry, C. Kiddle, N. Markatchev, R. Simmonds, T. Tan, M. Arlitt, and B. Walker. Facebook meets the virtualized enterprise. In 2008 12th International IEEE EDOCC.

[25] http://www.wikibench.eu/

[26] O. Sukwong, A. Sangpetch, H. S. Kim. SageShift: Managing SLAs for Highly Consolidated Cloud. In InfoCom 2012.

[27] http://www.nas.nasa.gov/publications/npb.html

[28] Kernel based virtual machine. www.linux-kvm.org/page/Main_Page

[29] http://www.eecs.harvard.edu/~chaki/bib/papers/jackson01maui.pdf

[30] G. Urdaneta, G. Pierre, M. van Steen. Wikipedia Workload Analysis for Decentralized Hosting. Elsevier Computer Networks, 2009.

[31] I. Goiri, R. Beauchea, K. Le, T. Nguyen, M. Haque, J. Guitart, J. Torres, R. Bianchini: GreenSlot: scheduling energy consumption in green datacenters. In SC 2011.

[32] K. Funaoka, S. Kato, and N. Yamasaki. 2008. Work-Conserving Optimal Real-Time Scheduling on Multiprocessors. In (ECRTS '08).

[33] M. Arlitt, C. Bash, S. Blagodurov, Y. Chen, T. Christian, D. Gmach, C. Hyser, N. Kumari, Z. Liu, M. Marwah, A. McReynolds, C. Patel, A. Shah, Z. Wang, and R. Zhou. Towards The Design And Operation Of Net-Zero Energy Data Centers. In ITherm 2012.

[34] N. Tolia, D. G. Andersen, and M. Satyanarayanan. Quantifying interactive user experience on thin clients. Computer, 39:46–52, 2006.

[35] J. Rolia, L. Cherkasova, M. Arlitt, A. Andrzejak, A Capacity Management Service for Resource Pools. In WOSP 2005, pp. 229–237.

[36] B. Lin, P. Dinda. VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling. In SC 2005.

[37] https://oss.oracle.com/ol6/docs/RELEASE-NOTES-UEK2-en.html

[38] http://www.kernel.org/doc/Documentation/cgroups/

[39] Y. Diao and A. Heching. Closed Loop Performance Management for Service Delivery Systems. In NOMS 2012.