Satisfying Service Level Objectives in a Self-Managing Resource Pool

Daniel Gmach, Jerry Rolia, and Lucy Cherkasova
Hewlett-Packard Laboratories
Palo Alto, CA, USA
`firstname.lastname@hp.com`

***Abstract*—We consider a self-managing, self-organizing pool of virtualized computer servers that provides infrastructure as a service (IaaS) for enterprise computing workloads. A global controller automatically manages the pool in a top down manner by periodically varying the number of servers used and re-assigning workloads to different servers. It aims to use as few servers as possible to minimize power usage while satisfying per-workload service level requirements. Each server is self-organizing. It has a local workload manager that dynamically varies the capacity allocated to each workload to satisfy per-workload service level objectives. This paper evaluates the impact of four alternative workload manager policies on the quality of service provided by the resource pool. The policies include: i) a non-work-conserving feedback controller, ii) a work-conserving feedback controller, iii) a work-conserving feedback controller with fixed per-workload scheduling weights to support differentiated service, and iv) a work-conserving feedback controller with dynamic per-workload weight to provide differentiated service while minimizing penalties. A case study involving three months of data for 138 SAP applications shows that the work-conserving policy significantly outperforms the non-work-conserving policy. The dynamic weight policy is better able to minimize penalties than the other policies while treating workloads fairly. Our study offers insights into the trade-offs between performance isolation, efficient resource sharing, and quality of service.**

***Keywords*-Resource Pool Management; Enterprise Workload Analysis; Differentiated Service; Quality of Service**

I. Introduction

Virtualization is gaining popularity in enterprise environments as a software-based solution for building shared hardware infrastructures. Forrester Research estimates that businesses generally only end up using between 8 and 20 percent of the server capacity they have purchased. Virtualization technology helps to achieve greater system utilization while lowering total cost of ownership and responding more effectively to changing business conditions. For large enterprises, virtualization offers a solution for server and application consolidation in shared resource pools. The consolidation of multiple servers and their workloads has an objective of minimizing the number of resources, e. g., computer servers, needed to support the workloads. In addition to reducing costs, this can also lead to lower peak and average power requirements. Lowering peak power usage may be important in some data centres if peak power cannot easily be increased.

Applications participating in consolidation scenarios can make complex demands on servers. For example, many enterprise applications operate continuously, have unique time-varying demands, and have performance-oriented Quality of Service (QoS) objectives. To evaluate which workloads can be consolidated to which servers, some preliminary performance and workload analysis should be done. In the simple naive case, a data centre operator may estimate the peak resource requirements of each workload and then evaluate the combined resource requirements of a group of workloads by using the sum of their peak demands. However, such an approach can lead to significant resource over-provisioning since it does not take into account the benefits of resource sharing for complementary workload patterns. In this work, to evaluate which workloads can be consolidated to which servers we employ a trace-based approach [1] that assesses permutations and combinations of workloads in order to determine a near optimal workload placement that provides specific quality of service.

The general idea behind trace-based methods is that historic traces offer a model of application demands that are representative of future application behaviour. Traces are used to decide how to consolidate workloads to servers. In our past work, we assumed that the placement of workloads would be adjusted infrequently, e. g., weekly or monthly [1]. However, by repeatedly applying the method at shorter timescales we can achieve further reductions in required capacity. In this work, we treat the trace-based approach as a placement controller that periodically causes workloads to migrate among servers to consolidate them while satisfying quality requirements. Such migrations [2] are possible without interrupting the execution of the corresponding applications. This approach provides a top-down form of self-management for the resource pool. It keeps the number of servers used in proportion to the time-varying demands of the workloads and in this way minimizes power usage.

In our study the workloads are not equally important, certain workloads have higher quality of service requirements than others. Each server must make best use of its resources to satisfy the requirements of its workloads. To achieve this, each server has a workload manager controller. The workload manager controller implements different *classes of service* (CoS). Higher CoS typically cost more for customers and provide for greater revenue to service providers.

However, service providers also incur greater penalties if a sufficient quality of service is not provided. Hence, workloads with a higher CoS can have a bigger impact on the service provider's profit than workloads with lower CoS. Of course, this impact needs to be reflected in the allocation of resources to maximize the service provider's profit. For quality of service we rely on a *compliance ratio* that is defined as the percentage of measurement intervals where all of a workload's demands are satisfied. We define a service level agreement where financial penalties are charged when a workload does not satisfy its expected compliance ratio.

This paper leverages our recent work [3]. In particular, we exploit the workload placement controller, simulation system, and case study input data. This paper differs in that we consider an integrated workload placement controller and workload manager. The focus of this paper is on policies that govern the workload manager. We consider non-work-conserving policies where workloads have allocations that are not shared with other workloads. We also consider work-conserving policies where available resources are always shared among workloads with pending demands.

The primary goal of this paper is to evaluate the impact of four workload manager policies on the quality of service provided by the resource pool. The policies include: i) a non-work-conserving feedback controller, ii) a work-conserving feedback controller, iii) a work-conserving feedback controller with fixed per-workload scheduling weights to support differentiated service, and iv) a work-conserving feedback controller with dynamic per-workload weights to provide differentiated service and maximize revenue. The fourth policy is self-organizing in that it dynamically adjusts the scheduling weights of workloads to achieve *sufficient* quality of service for each workload while minimizing financial penalties to the resource provider.

To assess the long term impact of such policies we exploit a host load simulation environment. The environment: models the placement of workloads on servers; simulates the competition for resources on servers; causes the controllers to execute according to a management policy; and dynamically adjusts the placement of workloads on servers. During this simulation process, the simulator collects metrics that are used to compare the effectiveness of the policies. The metrics include: the compliance ratio, which demonstrates the ability of the policies to support differentiated service; total financial penalties; and, additional metrics that compare total capacity used and other aspects of quality of service for the resource pool as a whole.

A case study involving three months of data for 138 SAP applications is used to evaluate the effectiveness of scheduler policies. We found that the work conserving policies significantly out-perform the non-work conserving policy and that the self-organizing policy is best able to minimize overall penalties while treating workloads fairly.

The remainder of this paper is organized as follows. Section II describes the workload placement and workload manager controllers, management policies, and metrics. The host load simulation environment is described in Section III. Section IV presents case study results. Section V describes related work. Finally, conclusions are offered in Section VI.

II. Controllers, Policies, and Quality Metrics

This section describes the workload placement and workload manager controllers, management policies, and quality metrics that are used to assess the effectiveness of management policies.

A. Workload Placement Controller

The workload placement controller has two components.

- *A simulator component* emulates the assignment of several application workloads on a single server. It traverses the per-workload time varying traces of historical demand to determine the peak of the aggregate demand for the combined workloads. If for each capacity attribute, e. g., CPU and memory, the peak demand is less than the capacity of the attribute for the server then the workloads fit on the server.
- *An optimizing search component* examines many alternative placements of workloads on servers and reports the best solution found. The optimizing search is based on a genetic algorithm [4].

The workload placement controller is based on the Capman tool that is described further in [1]. It supports both consolidation and load levelling exercises. Load levelling balances workloads across a set of resources to reduce the likelihood of service level violations. Capman supports the controlled overbooking of capacity that computes a required capacity for workloads on a server that may be less than the peak of aggregate demand. It is capable of supporting a different quality of service for each workload [5]. Without loss of generality, this paper considers the highest quality of service, which corresponds to a required capacity for workloads on a server that is the peak of their aggregate demand. For this paper, the workload placement controller operates at regular timed intervals, e.g., every four hours. As will be shown, this is frequent enough to exploit time of day variations in workload demands.

B. Workload Manager Controller

The purpose of a workload manager controller is to periodically adjust the resource allocation for each workload on a server. This is done at short timescales to adapt to short term fluctuations in workload demands. Figure 1 illustrates the relationship between virtual machines, a physical server, and a workload management controller.

The left box of Figure 1 denotes a physical server, which is hosting several virtual machines. We assume each workload is associated with its own virtual machine. The figure shows that physical servers provide a monitoring service and

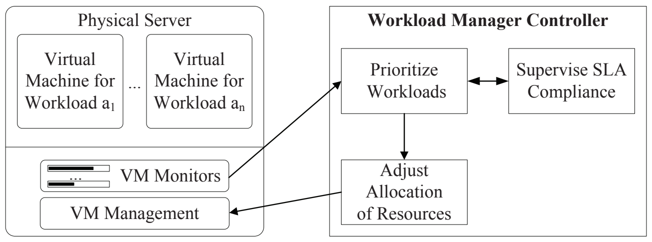

Physical Server
Virtual Machine for Workload a_1
...
Virtual Machine for Workload a_n
VM Monitors
VM Management
Workload Manager Controller
Prioritize Workloads
Supervise SLA Compliance
Adjust Allocation of Resources

Figure 1. Architecture of the Workload Management Service

a VM management interface. The VM management interface enables the resource allocation of virtual machines to be adjusted at runtime.

The right box of Figure 1 shows the architecture of the workload management controller. Once per workload manager control interval, the workload manager controller communicates with VM monitors and the VM management interface on the physical server. VM monitors provide access to recent resource demands, utilization values, and service level metrics. The resource allocation for VMs is adjusted via the VM management interface.

We consider a VM management interface that supports the Xen credit scheduler [6]. It manages a VM's CPU weight, CPU cap, and physical memory. A VM that is competing for CPU resources receives a service rate in proportion to its weight divided by the total weights of all workloads that are competing for CPU resources. A CPU cap limits the total allocation that a VM can receive per control interval. The following subsections describe the workload manager policies we use in more detail.

1) Policies i) and ii): Non-Work Conserving vs. Work-Conserving: Policy i) *nwc demand* is non-work-conserving. With policy i), the workload manager decides a CPU cap for each workload for the next control interval based on the workload's recent CPU demands. The greater the recent demands, the greater the CPU cap. The value of the CPU caps are scaled to use all of the CPU capacity on a server. The caps are scaled up when not all capacity is needed, and scaled down if the server is oversubscribed. Weights are not used. Each workload is entitled to receive up to its CPU cap in CPU capacity during the control interval. If it doesn't use the capacity then the capacity is not available to any other workload. Historically, non-work conserving policies have been used to provide for better performance isolation between workloads that share a server. For example, in some cases it is important to make sure that one workload does not become dependent on capacity that is being paid for by other workloads.

Policy ii) *wc demand* is work-conserving. With policy ii) the computed CPU caps are treated as weights and are not enforced as hard capacity limits. In this way unused capacity is always available to workloads that need it. The case study evaluates the impact of non-work-conserving versus work-conserving policies in shared resource pool environments.

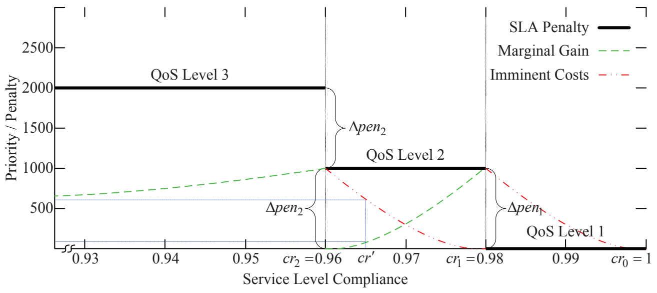


Figure 2. Dynamic Weights for Workloads

For policies i) and ii), classes of service are supported in the following way. If a server is oversubscribed, then the CPU caps are allocated to workloads of the highest CoS first, then to the next CoS, and so forth. Lowest CoS workloads suffer when a server is oversubscribed.

2) Policy iii) and Policy iv): CoS based Weights and Dynamic Weights: Policy iii) *wc static* is work-conserving. The workload management controller associates each CoS with a different weight. The higher the CoS the greater the weight. All workloads with the same CoS have the same weight. Policy iii) is only considered in a work-conserving mode. Workload management services based on static weights for workloads tend to over-provision capacity to high CoS workloads. Consequently, workloads with low CoS suffer from resource deficits resulting in bad QoS and accrued penalties for these workloads.

Policy iv) *wc dynamic* is work-conserving. It is an approach for dynamically adjusting workload weights for multiple workloads to better satisfy the needs of all workloads. The approach relies on a Service Level Agreement (SLA) for one or more CoS. Each CoS has a desired compliance ratio cr and a step function that defines penalties that increase as the compliance ratio achieved for a workload diminishes. We refer to the steps of compliance ratio intervals as *QoS levels*.

The dynamically calculated weight for a workload considers two different economic aspects: *imminent costs* and *marginal gain*. Imminent costs model the risk of missing the current QoS level. The closer the resource compliance ratio gets to the next lower QoS level, the higher the probability that the workload quality will drop to the next lower QoS level. The imminent cost function expresses this probability. It is defined using a decreasing, convex polynomial function of degree deg that is fit to equal the penalties Δpen_i for missing the current QoS level i at its lower boundary and a penalty of 0 at its upper boundary. The marginal gain function is modelling the chance to achieve the next higher QoS level. If the next higher QoS level comes into reach, the weight of the workload increases. The marginal gain function and the imminent cost function are defined similarly, but the marginal gain function uses an increasing, convex polynomial function. The weight of a workload is computed for each control interval as the maximum of imminent costs and marginal gains. Each workload can have its own distinct functions. The resulting weights are used to prioritize the workloads.

Figure 2 illustrates the step function for an SLA with one CoS, a desired compliance ratio of 98% and where a penalty of \$1000 is due for every 2 percentage points below the desired satisfaction metric. The maximum penalty is limited to \$2000. No penalties are due if at the end of the evaluation period more than 98% of all intervals have had their demands satisfied. \$1000 is due if between 96% and 98% of intervals had their demands satisfied, and \$2000 is due if less than 96% of intervals had their demands satisfied.

Figure 2 shows a current service level compliance $cr' = 0.965$. Consequently, the workload currently just fulfills QoS level 2. The imminent costs are indicated by the decreasing polynomial functions. Given a compliance ratio $cr' = 0.965$ and utility functions with polynomial degree 2, the imminent costs of the workload are approximately 560. In the figure, the current compliance ratio cr' of the workload is comparatively far away from the next higher boundary cr_1 resulting in a low marginal gain of about 60. The current weight of the workload is computed as $\max\{560, 60\} = 560$.

The degrees of the imminent costs and the marginal gain functions constitute the dynamic and static influence on weight. A high degree strengthens the dynamic influence and a low degree weakens it. In case of a very high degree, the compliance ratios of all workloads tend to approach the next lower QoS threshold closely. Hence, the risk of falling into the next lower QoS level is rather high for all workloads, including the important ones. If the polynomial degree deg is small, high priority workloads tend to keep some distance to their next lower compliance thresholds and if deg even approximates zero, the dynamic weight approach approximates the static weight approach.

C. Efficiency and Quality Metrics

To compare the long term impact of management policies we consider several additional metrics. These include:

- total server CPU hours used and total server CPU hours idle;
- normalized server CPU hours used and normalized server CPU hours idle;
- minimum and maximum number of servers;
- CPU resource access quality per hour; and
- the number of migrations per hour.

The total server CPU hours used corresponds to the sum of the per workload demands. Total server CPU hours idle is the sum of idle CPU hours for servers that have workloads assigned to them. The server CPU hours idle shows how much CPU capacity is not used on the active servers. Normalized values are defined with respect to the total demand of the workloads as specified in the workload demand traces. Note that if normalized server CPU hours used is equal to 1 and normalized server CPU hours idle are equal to 1.5 then this corresponds to an average CPU utilization of 40%.

The minimum and maximum number of servers for a policy are used to compare the overall impact of a management policy on capacity needed for server infrastructure. This determines the cost of the infrastructure.

We define a quality metric named *violation penalty* that is based on the number of successive intervals where a workload's demands are not fully satisfied and the expected impact on the customer. Longer epochs of unsatisfied demand incur greater penalty values, as they are more likely to be perceived by those using applications. For example, if service performance is degraded for up to 5 minutes customers would start to notice. If the service is degraded for more than 5 minutes then customers may start to call the service provider and complain. Furthermore, larger degradations in service must cause greater penalties.

The quality of the delivered service depends on how much the service is degraded. If demands greatly exceed allocated resources then the utility of the service suffers more than if demands are almost satisfied. Thus, for each violation a penalty weight w_{pen} is defined that is based on the expected impact of the degraded quality on the customer. The violation penalty value pen for a violation with I successive overloaded measurement intervals is defined as $pen = I^2 \max_{i=1}^{I} (w_{pen,i})$, where $w_{pen,i}$ is the penalty in the ith interval. Thus longer violations tend to have greater penalties than shorter violations[1]. The weight function used for CPU is given below. The sum of penalty values over all workloads over all violations defines the violation penalty for the metric.

For CPU allocations, we estimate the impact of degraded service on a customer using a heuristic that compares the actual and desired utilization of allocation for the customer. An estimate is needed because we do not have measurements that reflect the actual impact on a customer. Let u_a and $u_d < 1$ be the actual and desired CPU utilization of allocation for an interval. If $u_a \leq u_d$ then we define the weight for the CPU penalty w_{pen}^{CPU} as $w_{pen}^{CPU} = 0$ since there is no violation. If $u_a > u_d$ then response times will be higher than planned so we must estimate the impact of the degradation on the customer. We define:

$$w_{pen}^{CPU} = 1 - \frac{1 - u_a^k}{1 - u_d^k}.$$

The penalty has a value between 0 and 1 and is larger for bigger differences and higher utilizations. The superscript k denotes the number of CPUs on the server. This formula is motivated by a formula that estimates the mean response time for the $M/M/k$ queue [7], namely $r = 1/(1 - u^k)$

[1]We note that such penalties may be translated to monetary penalties in financially driven systems and that monetary penalties are likely to be bounded in such systems.

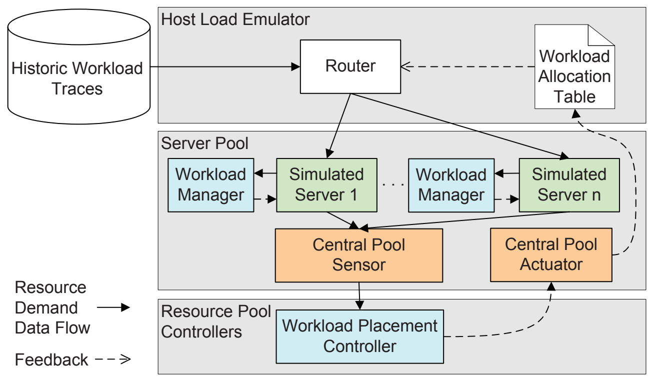


Figure 3. Architecture of the Host Load Emulator

estimates the mean response time for a queue with k processors and unit service demand [8]. The power term k reflects the fact that a server with more processors can sustain higher utilizations without impacting customer response times. Similarly, a customer that has a higher than desired utilization of allocation will be less impacted on a system with more processors than one with fewer processors.

To summarize, the CPU penalty reflects two factors. These are the length of the violation and the severity of the violation which is captured by a weight function.

III. Host Load Emulator

Predicting the long term impact of integrated management policies for realistic workloads is a challenging task. We employ a flexible host load emulation environment to evaluate many management policies for resource pools in a time effective manner.

The architecture of the host load emulation environment is illustrated in Figure 3. The emulator takes as input historical workload demand traces, an initial workload placement, server resource capacity descriptions, and a management policy. The server descriptions include numbers of processors, processor speeds, real memory size, and network bandwidth. A routing table directs each workload's historical time varying resource requirement data to the appropriate simulated server. Each simulated server has a workload manager that maintains the local resource allocation. Using one of the four workload manager policies, the simulated server determines how much of the workload demand is and is not satisfied. The central pool sensor makes time varying information about satisfied demands available to resource pool management controllers via an open interface. The interface also is used to integrate different controllers with the emulator without recompiling its code.

Controllers periodically gather accumulated metrics and make decisions about whether to cause workloads to migrate from one server to another. Migration is initiated by a call from a controller to the central pool actuator. In our emulation environment this causes a change to the routing table that reflects the impact of the migration in the next simulated time interval. Furthermore, for each workload that migrates, a CPU overhead is added to the source and destination servers. The overhead is proportional to the estimated transfer time based on the memory size of the virtual machine and the network interface card bandwidth. The migration overhead is described in detail in [3].

During the emulation process the metrics defined in Section II-C are gathered. Different controller policies cause different behaviours that we observe through these metrics.

IV. Case Study

This section evaluates the effectiveness of the proposed management policies using three months of real-world workload demand traces for 138 SAP enterprise applications. The traces are obtained from a data centre that specializes in hosting enterprise applications such as customer relationship management applications for small and medium sized businesses. Each workload was hosted on its own server so we use resource demand measurements for a server to characterize the workload's demand trace. The measurements were originally recorded using *vmstat* [9]. Traces capture average CPU and memory usage as recorded every 5 minutes.

As many of the workloads are interactive enterprise workloads, a maximum utilization of 0.66 is desired to ensure interactive responsiveness. Hence, CPU demands in the historical workload traces are scaled with a factor of 1.5 to achieve a target utilization of 0.66. The resource pool simulator operates on this data walking forward in successive 5 minute intervals. In addition to the three months of the real-world demand traces we used data from the previous month to initialize the demand buffers of the central pool sensor. This enables the integrated management services to access prior demand values at the start of a simulation run.

We consider the following resource pool configuration [2]: each server consists of 8 x 2.93-GHz processor cores, 128 GB of memory, and two dual 10 Gb/s Ethernet network interface cards for network traffic and virtualization management traffic, respectively.

Section IV-A gives a workload characterization for the SAP workloads considered in the study. Section IV-B considers the question: how much capacity and hence power can be saved by periodically consolidating workloads? The section assumes perfect knowledge about future workload demands and its results give a baseline for capacity savings that is used for the rest of the case study. Sections IV-C and IV-D do not assume perfect knowledge of future demands. They consider the integration of the workload placement controller and per server workload manager controllers and their effectiveness at supporting QoS.

[2]Service providers can use the proposed approach for evaluating different hardware platforms. For example, in [3] we made recommendations regarding server and blade based resource pool configurations.

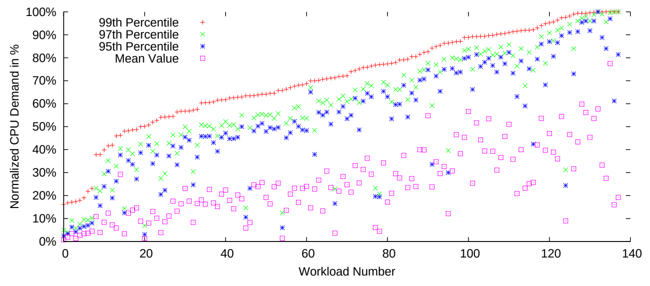


Figure 4. Top Percentile of CPU Demand for Applications under Study

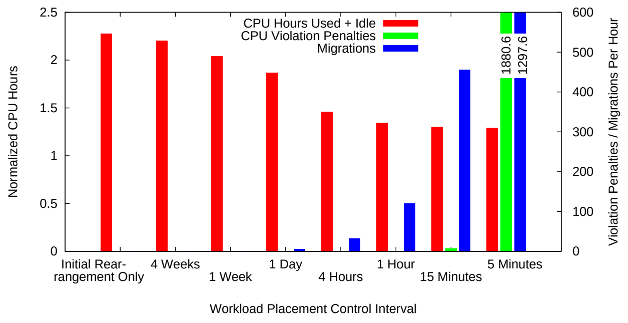


Figure 5. Simulation Results Assuming Perfect Knowledge.

A. *Workload Characteristics*

The use of virtualization technology enables the creation of shared server pools where multiple application workloads share each server in the pool. Understanding the nature of enterprise workloads is crucial to properly design and provision current and future services in such pools.

Existing studies of Internet and media workloads [10], [11] indicate that client demands are highly variable ("peak-to-mean" ratios may be an order of magnitude or more), and that it is not economical to overprovision the system using "peak" demands. We present results that illustrate the peak-to-mean behaviour for 138 enterprise application workloads. Figure 4 gives the percentiles of CPU demand for the 138 applications over the period of four months. The illustrated demands are normalized as a percentage with respect to their peak values. Several curves are shown that illustrate the 99th, 97th, and 95th percentile of demand as well as the mean demand. The workloads are ordered by the 99th percentile for clarity. The figure shows that more than half of all studied workloads have a small percentage of points that are very large with respect to their remaining demands. The left-most 60 workloads have their top 3% of demand values between 10 and 2 times higher than the remaining demands in the trace. Furthermore, more than half of the workloads observe a mean demand less than 30% of the peak demand. These curves show the bursty nature of demands for most of the enterprise applications under study. Consolidating such bursty workloads onto a smaller number of more powerful servers is likely to reduce the CPU capacity needed to support the workloads.

B. *Performance, Quality, and Power Assuming Perfect Knowledge*

In this section, we consider an *ideal* workload placement strategy. This approach assumes that we have perfect knowledge of future resource demands. It gives an upper bound for the potential capacity savings from consolidating workloads at different time scales. We use this bound later in the paper to determine how well our policies, that do not have perfect knowledge, perform compared to the ideal case.

Figure 5 shows the results of an emulation where we use the workload placement controller to periodically consolidate the 138 workloads to a small number of servers in the resource pool. For this scenario, for a given time period, the workload placement controller chooses a placement such that each server is able to satisfy the peak of its workload CPU and memory demands. We note however that the workload placement controller does not take into account migration overheads. Migration overheads can cause CPU violation penalties. The workload manager implements policy iii), i. e., *wc static* with equal weights for the workloads. The figure shows the impact on capacity requirements of using the workload placement controller once at the start of the three months, and for cases with a control interval of 4 weeks, 1 week, 1 day, 4 hours, 1 hour, and 15 minutes. The figure shows that re-allocating workloads every 4 hours captures most of the capacity savings that can be achieved, i. e., with respect to reallocation every 15 minutes. The 4 hour and 15 minute scenarios required a peak of 19 servers. All the other scenarios also had peaks between 19 and 21 servers. The minimum number of servers for the 4 hour scenario was 9. All the other scenarios with longer intervals had minimums between 16 and 21. For the 4 hour scenario, we note that the total server CPU hours used were 149114, which is more than twice of the idle CPU hours (67750), giving an average utilization close to 69% over the three month period with a negligible hourly CPU violation penalty value of 0.4.

The figure also shows that as expected as the control interval drops to the hourly, fifteen minute, and five minute levels the number of migrations per hour increases proportionally as most workloads are likely to be reassigned. The resulting migration overheads increase the CPU quality violations. Table I gives a more detailed breakdown of the violations for the 4 hour control interval case.

Interval Duration	Total Number	Average Number
5 Minute	1090	13 per Day
10 Minute	161	1.9 per Day
15 Minute	14	1.2 per Week
20 Minute	3	1 per Month

Table I
CPU QUALITY VIOLATIONS ASSUMING PERFECT KNOWLEDGE FOR THE 4 HOUR CONTROL INTERVAL.

In later subsections, we treat the results from the 4 hour ideal case as the baseline for capacity and quality and evaluate how much of these *ideal* advantages we are able to achieve in practice without assuming perfect knowledge. The workload placement controller control interval is chosen as four hours.

C. Workload Management

We now consider the effectiveness of integrating a global workload placement controller and per server workload manager controllers in real scenarios. For workload placement, we employ a *Historical* policy. This policy uses historical data from the previous three weeks as a predictor of the future workload demands. It performed best in a previous comprehensive study on workload placement controller policies [3]. Using this policy, the workload placement controller required only 18 percent more capacity than the ideal case.

The workload placement controller determines the required capacity and calculates a workload placement. Within the 4 hour control intervals workload managers allocate the available per server capacity to their workloads. Of course, from time to time the workload placement controller underestimates capacity resulting in resource bottlenecks. During these shortages, the workload manager needs to allocate resources to the workloads in such a way that provides the best quality of service.

Our study evaluates the effectiveness of the four policies described in Section 2.2. For policy iv) we employ dynamic utility functions with polynomial degree 2 and 6, referred to as *wc dynamic degree 2* and *wc dynamic degree 6*, respectively.

Finally, to illustrate the trade-off between capacity and quality, we introduce a per server headroom during the workload placement process. The *headroom* leaves a certain fraction of the server unallocated during the workload placement controllers placement exercises. The greater the headroom the less likely that aggregate demands will exceed the capacity of the server. In our experiments, the headroom is varied from 0% to 20% in 5% steps. For our study, the minimum and maximum numbers of required servers is determined by the workload placement controller. However, the non-work-conserving policy impacts the maximum number of required servers as it flattens peak demands as seen as inputs for the workload placement controller. Depending on the headroom, the minimum number of servers was either 14 or 15 and the maximum ranged from 19 to 24.

Figure 6 shows the quality and the total required capacity for the different experiments assuming one Class of Service (CoS). Note that the y-axis of the figure has a logarithmic scale. The figure shows that the work-conserving policies achieve much better overall quality. When using a similar amount of capacity the incurred quality penalties are about 10 times less. Hence, we strongly encourage to use work-conserving policies in shared resource pools. The overall

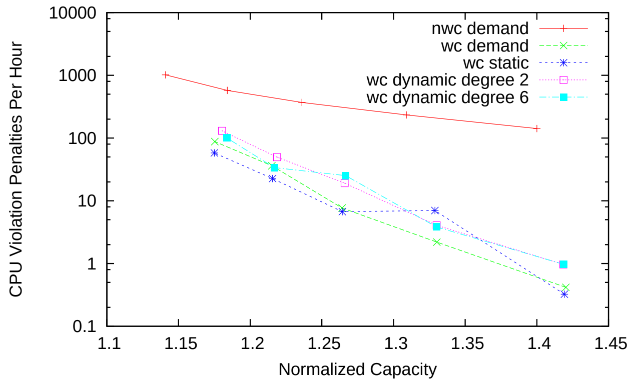


Figure 6. Integrated WP and WM Assuming 1 CoS

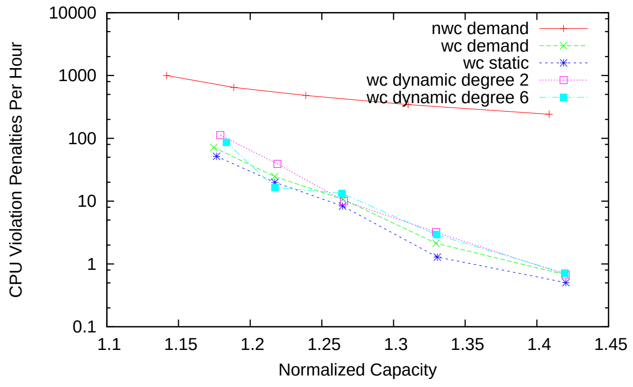


Figure 7. Integrated WP and WM Assuming 3 CoS

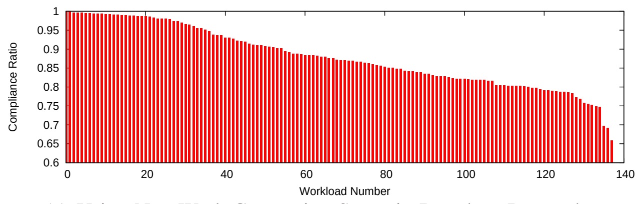
quality and required capacity were very similar for all the work-conserving policies.

Figure 7 shows the corresponding results for a scenario with three CoS that have 30 gold workloads, 40 silver workloads, and 68 bronze workloads. A comparison of the figures shows that providing differentiated QoS does not impact overall quality as measured with respect to the resource pool. In the next section, we evaluate how well the different workload manager policies provide differentiated quality of service to the workloads.

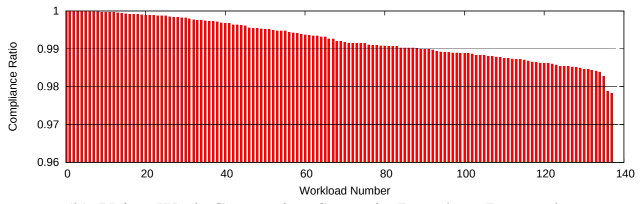
D. Providing Differentiated QoS

The previous section evaluated the impact of the different workload manager policies on overall capacity and quality. Now, we consider per-workload quality for the different policies. We consider the experiments from the previous section where the workload placement controller did not employ a headroom, i. e., the headroom was 0%. These experiments challenge the local workload managers most. We consider one and three CoS scenarios using the SLA definition from Section II-B.

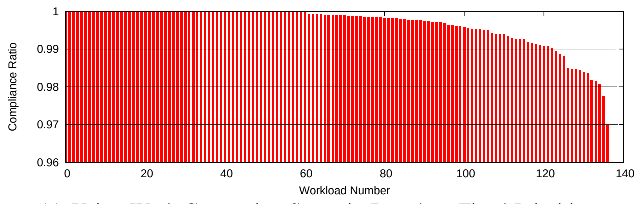
For the one CoS scenario, the SLA is defined by a compliance ratio of 99% and a penalty of \$100 per percentage of missed compliance. Figure 8 shows the achieved per-workload compliance ratios for the different policies. Figure 8(a) shows that policy i), i. e., *nwc demand*, offers the worst quality. For more than half of the workloads the resource demands are not fully satisfied in more than 10%



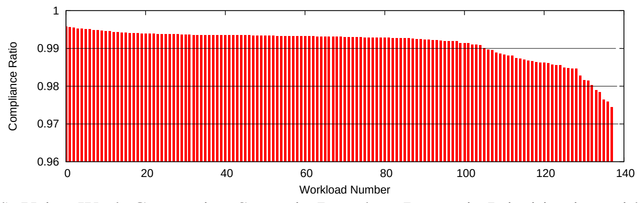
(a) Using Non-Work-Conserving Scenario Based on Demands



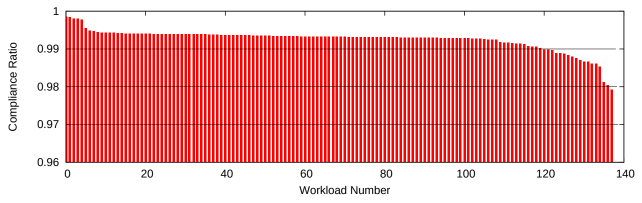
(b) Using Work-Conserving Scenario Based on Demands



(c) Using Work-Conserving Scenario Based on Fixed Priorities



(d) Using Work-Conserving Scenario Based on Dynamic Prioritization with Polynomial Degree 2



(e) Using Work-Conserving Scenario Based on Dynamic Prioritization with Polynomial Degree 6

Figure 8. Achieved Compliance Ratio *cr* for Each Workload when Allocating Server to 100%

of the simulation intervals. In a real production system this would be not acceptable and the service provider would need to provide more capacity to meet its service level objectives. All the other policies offer a compliance ratio greater than 97% for all workloads.

The work-conserving policy, policy ii), i. e., *wc demand*, shown in Figure 8(b) dramatically improves the per-workload quality even though it does not consider the compliance ratio. With the SLA we defined, the sum of the per-workload penalties is $5100.

The results of the static policy, policy iii), i. e., *wc static*, are shown in (Figure 8(c)). It further reduces the sum of penalties to $2100. Penalties are reduced because the weight based policy favours smaller workloads as compared to the

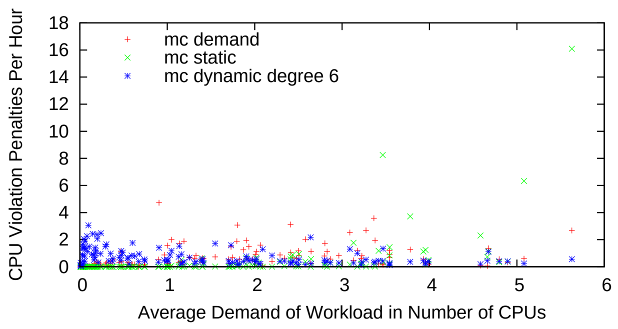


Figure 9. CPU Violation Penalties per Workload With Regard to Average Demand

demand based policy.

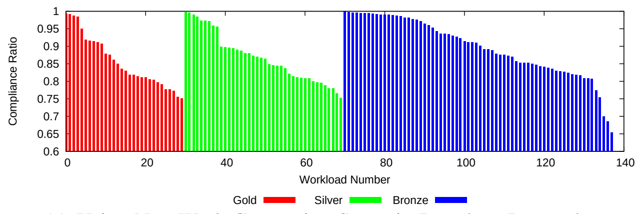
The results of policy iv), i.e., *wc dynamic degree 2* and *wc dynamic degree 6*, are shown in Figure 8(d) and 8(e), respectively. Policy iv) considers the workloads' current compliance ratios and allocates capacity in a manner that aims to minimize penalties. The degree of the policy function determines the aggressiveness of the dynamic aspect, i. e., how close the workload manager lets the workloads get to their next lower threshold. The quadratic polynomial function incurred in total $3700 in penalties. Policy *dynamic wc degree 6* is more aggressive than the degree 2 approach and reduces penalties to $1900. *wc dynamic degree 6* has the smallest penalties for one class of service scenario.

Figure 9 shows the CPU violation penalties per workload with regard to the average demand for policy ii), iii), and iv). Policy iii), i. e., *wc static*, degrades a few large workloads. This frees enough capacity to fully satisfy demands of many smaller workloads. But this behaviour is not desirable in shared resource pools. The figure shows that the demand based and the dynamic policies both allocate resources fairly to workloads regardless of their size.

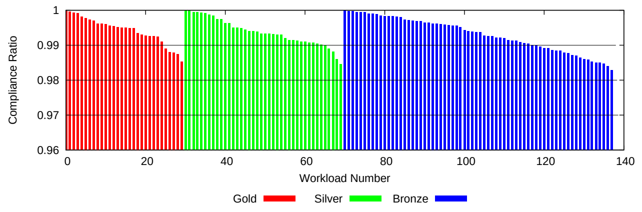
Next, we consider three classes of service: gold, silver and bronze. All three CoS have compliance ratio requirements of 99% but with penalties of $400, $200, and $100 per percentage missed, respectively.

Figure 10 shows the corresponding results. As in the one CoS case, Policy i) *nwc demand* resulted in unacceptable quality. Policy ii), *wc demand*, as shown in Figure 10(b) chooses weights such that each workload gets the same fraction of its demand satisfied but does not consider CoS and compliance ratios. As expected, the per-workload compliance ratios are similarly for each class of workloads, leading to total penalties of $5000. The gold class alone incurred penalties of $2000.

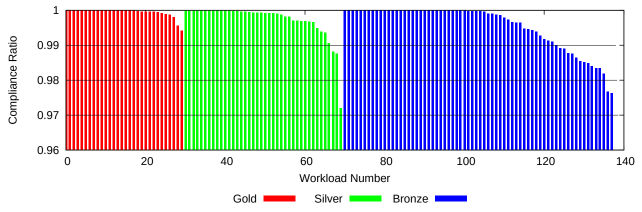
wc static supports differentiated service by offering each workload a weight that is in proportion to the penalty of its CoS. In this study the weights for gold, silver and bronze are 4, 2, and 1, respectively. Figure 10(c) shows that *wc static* improves compliance ratios for the higher CoS. No gold workloads fall below the desired 99% level. However, the improved quality of gold workloads is achieved through



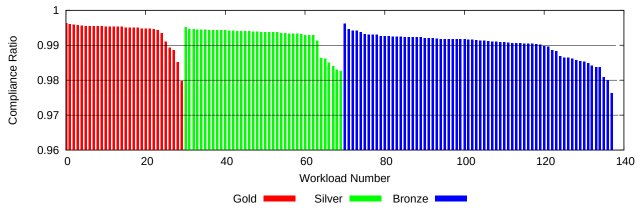
(a) Using Non-Work-Conserving Scenario Based on Demands



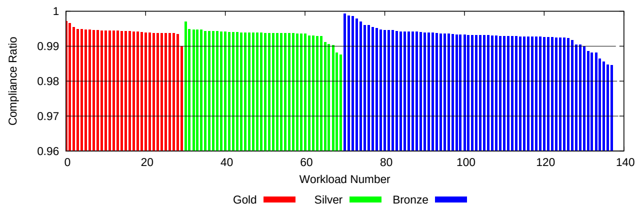
(b) Using Work-Conserving Scenario Based on Demands



(c) Using Work-Conserving Scenario Based on Fixed Priorities



(d) Using Work-Conserving Scenario Based on Dynamic Prioritization with Polynomial Degree 2



(e) Using Work-Conserving Scenario Based on Dynamic Prioritization with Polynomial Degree 6

Figure 10. Achieved Compliance Ratio *cr* for Each Workload when Allocating Server to 100%

the degradation of many lower prioritized workloads. The figure shows that 2 bronze workloads only achieved the 97% compliance ratio and 13 more only a 98% level. Bronze workloads alone incurred penalty costs of $1700. Also one silver workload even missed the 98% ratio. The total penalties for all workloads sum up to $2500. This is a reduction of 50% to compared to *wc demand*.

wc dynamic degree 2 and *wc dynamic degree 6* dynamically alter the weights associated with each workload to minimize penalties. Figure 10(d) shows the achieved compliance ratios for the quadratic polynomial function, which incurred total penalties of $5200. However, workloads do not overachieve their SLA levels compared to the static prioritization scenario. This helps to improve the resource access quality for bronze workloads. Now all except two bronze workloads obtain the 98% level but 16 bronze workloads still fall below a 99% compliance ratio.

Figure 10(e) shows the results for *wc dynamic degree 6*, i.e., a degree 6 polynomial. The gold and silver workloads less overachieve the 99% compliance ratio. This helps low priority workloads to improve their quality. Now, only one gold, one silver and 8 bronze workloads miss the desired 99% compliance ratio level. The total penalties for all workloads are $1600, which is 68% less than policy ii) *wc demand* and 36% less than policy iii) *wc static*.

To summarize, policy iv), i.e., *wc dynamic degree 6*, provides for the lowest penalties for both the one and three CoS cases. The next best policy was policy iii) *wc static* but we note that it favours smaller workloads, which is not desirable in real production environments.

V. Related Work

Server consolidation is becoming an increasingly popular approach in enterprise environments to better utilize and manage systems. The problem of efficient workload placement and workload management in such environments is in a centre of attention for many research and product groups.

In our work, we chose to represent application behaviour via workload demand traces. Many research groups have used a similar approach to characterize application behaviour and applied trace-based methods to support what-if analysis in the assignment of workloads to consolidated servers [12], [5], [13], [14], [15], [1], [16]. A consolidation analysis presented in [12] packs existing server workloads onto a smaller number of servers using an Integer Linear Programming based bin-packing method. Unfortunately, the bin-packing method is NP-complete for this problem. This makes the method impractical for larger consolidation exercises and on-going capacity management. There are now commercial tools [17], [18], [19], [20] that employ trace-based methods to support server consolidation exercises, load balancing, ongoing capacity planning, and simulating placement of application workloads to help IT administrators improve server utilization. We believe the workload placement service we employ has advantages over other workload placement services described above. It addresses issues including classes of service and placement constraints. The approach is also able to minimize migrations over successive control intervals. Some researchers propose to limit the capacity requirement of an application workload to a percentile of its demand [14]. This does not take into account the impact of sustained performance degradation over time on user experience as our required capacity definition does. Others look only at objectives for resources as a whole [16] rather than making it possible for each workload to have an independently specified objective.

Earlier work at HP Labs focussed on capacity planning and workload placement [1], [12], [21]. This work has assumed the presence of allocation and arbitration mechanisms of the kind that exist in industrial products [22], [23], [24], [25]. These products implement control loops and are motivated by methods from control theory. Such products support demand based allocations, allocations based on workload response times or throughputs, priorities, entitlements, and other features. However the workload managers typically have control parameters that must be specified or configured in an off-line manner on a per-workload basis. More recently, adaptive controller technologies have been applied to allocation and arbitration [26], [27] for such control loops. For example, Wang et. al. [27] use an adaptive controller to dynamically adjust the gain parameters of an integral controller based workload manager to better meet application level response time objectives. All these papers mentioned above assume non-work conserving CPU scheduler and deal with the capped mode. However, there are significant benefits for application performance when a work-conserving mode is used instead as shown in [28]. In our work, we compare achievable performance QoS when using work-conserving versus non-work-conserving schedulers as well as different share allocation strategies.

Recently, virtualization platforms such as VMware and Xen [2], [29] provide the ability to dynamically migrate VMs from one physical machine to another without interrupting application execution. Wood et al. [30] present Sandpiper, a system that automates the task of monitoring virtual machine performance, detecting hotspots, and initiating any necessary migrations. Sandpiper implements heuristic algorithms to determine which virtual machine to migrate from an overloaded server, where to migrate it, and a resource allocation for the virtual machine on the target server. Sandpiper implements a black-box approach that is fully OS- and application-agnostic and a gray-box approach that exploits OS- and application-level statistics. In our work, we use virtual machine migration as instructed by a workload placement controller to optimize the global workload placement according to the observed load patterns. 1000 Islands Project [31] aims to provide an integrated capacity and workload management for the next generation data centres. In the paper, the authors evaluate one loose integration policy for different controllers, while our paper provides a detailed performance study evaluating outcome of the integration policies at the resource pool and the node level and uses a set of novel QoS metrics. Our paper also considers the integration of a per-server workload manager and uses real system traces for this purpose whereas paper [31] does not.

There are many related works on policy-based management. For example, in [32], the authors statically derive and then dynamically refine low-level service level specifications to meet given SLAs while maximizing business profit.

VI. Conclusions and Future Work

This paper evaluates the impact of different policies for managing a shared resource pool providing infrastructure as a service to enterprise applications. The resource management system has two levels of controllers. The first level is a workload placement controller that periodically consolidates workloads to an appropriate number of servers in the pool. Its role is to keep the number of servers used in line with the aggregate demands of workloads and in that way minimize power usage. The second level of controllers are per-server workload manager controllers. A server's workload manager controller dynamically allocates capacity to workloads assigned the server.

We consider the impact of four different policies for the workload manager. First we compare the effectiveness of a non-work conserving policy with a work conserving policy and find that the work-conserving policy significantly out performs the non-work conserving policy. This is an important result given that much of the recent literature promotes the use of non-work conserving controllers. Next, we consider policies that either statically or dynamically assign weights to different workloads. We found that the dynamic approach performed best. It was able to reduce the penalties incurred in our study the most while treating workloads fairly. However, it is important to use an appropriate function to guide the choice of weights.

Our future work includes evaluating other instances of controllers and management policies, and to develop management policies that react well to more kinds of workloads and different kinds of simulated failures. Our compliance ratio could be enhanced to take into account bursts of degraded service. We also plan to consider the impact of service agreement windows, where utility and quality of service are assessed at regular intervals. Further, we plan to improve the utility functions by exploiting expected workload behaviour. Finally, we also plan to consider a greater variety of workloads.

References

[1] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak, "A Capacity Management Service for Resource Pools," in *Proc. of the 5th Int. Workshop on Software and Performance (WOSP)*, Palma, Illes Balears, Spain, 2005, pp. 229–237.

[2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live Migration of Virtual Machines," in *Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005, pp. 273–286.

[3] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics," in *Proc. of the 38th IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, Anchorage, Alaska, USA, Jul. 2008.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975.

[5] L. Cherkasova and J. Rolia, "R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools," in *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN)*, Philadelphia, USA, 2006.

[6] Xen Wiki, "Credit Scheduler: Credit-Based CPU Scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler, 2007.

[7] L. Kleinrock, *Queueing Systems, Volume 1: Theorie*. New York, USA: John Wiley & Sons, 1975.

[8] J. Rolia, "Predicting the Performance of Software Systems," Ph.D. dissertation, University of Toronto, 1992.

[9] H. Ware and F. Frdrick, "Linux Man Page: vmstat(8)," http://linux.die.net/man/8/vmstat, 1994.

[10] M. F. Arlitt and C. L. Williamson, "Web Server Workload Characterization: The Search for Invariants," in *Proc. of the ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems*. Philadelphia, PA, USA: ACM, May 1996, pp. 126–137.

[11] L. Cherkasova and M. Gupta, "Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads," in *Proc. of the 12th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. New York, NY, USA: ACM, 2002, pp. 33–42.

[12] A. Andrzejak, , M. Arlitt, and J. Rolia, "Bounding the Resource Savings of Utility Computing Models," HP Labs, Tech. Rep. HPL-2002-339, 2002.

[13] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper, "Adaptive Quality of Service Management for Enterprise Services," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 1, 2008.

[14] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource overbooking and application profiling in shared hosting platforms," *ACM SIGOPS Operating System Review*, vol. 36, Special Issue: Cluster Resource Management,, pp. 239–254, 2002.

[15] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, "Statistical Service Assurances for Applications in Utility Grid Environments," *Performance Evaluation*, vol. 58, no. 2+3, pp. 319–339, 2004.

[16] S. Seltzsam, D. Gmach, S. Krompass, and A. Kemper, "AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications," in *Proc. of the 22nd Int. Conf. on Data Engineering (ICDE), Industrial Track*, Atlanta, Georgia, USA, Apr. 2006.

[17] VMware, "VMWare Capacity Planner," http://www.vmware.com/products/capacity_planner/, 2008.

[18] HP, "HP Integrity Essentials Capacity Advisor," http://h71036.www7.hp.com/enterprise/cache/262379-0-0-0-121.html, 2008.

[19] IBM, "Tivoli Performance Analyzer," http://www.ibm.com/software/tivoli/products/performance-analyzer/, 2008.

[20] TeamQuest, "TeamQuest – IT Service Optimization," http://www.teamQuest.com, 2008.

[21] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating Enterprise Application Placement in Resource Utilities," in *Proc. of the 14th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM)*, Heidelberg, Germany, 2003, pp. 118–129.

[22] HP PRM, "HP Process Resource Manager," http://docs.hp.com/en/B8733-90025/index.html, 2009.

[23] HP-UX WLM, "HP-UX Workload Manager," http://h20338.www2.hp.com/hpux11i/cache/328328-0-0-0-121.html, 2008.

[24] IBM, "IBM Application Workload Manager," http://www.ibm.com/servers/eserver/xseries/systems_management/director_4/awm.html, 2009.

[25] IBM, "IBM Enterprise Workload Manager," http://www.ibm.com/developerworks/autonomic/ewlm/, 2009.

[26] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. New York, NY, USA: John Wiley & Sons, 2004.

[27] Z. Wang, X. Zhu, and S. Singhal, "Utilization vs. SLO-Based Control for Dynamic Sizing of Resource Partitions," in *Proc. of the 16th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM)*, Barcelona, Spain, 2005, pp. 133–144.

[28] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the Three CPU Schedulers in Xen," *ACM SIGMETRICS Performance Evaluation Review (PER)*, vol. 35, no. 2, pp. 42–51, 2007.

[29] VMware, "VMware VMotion," http://www.vmware.com/products/vi/vc/vmotion.html, 2008.

[30] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box Strategies for Virtual Machine Migration," in *Proc. of the 4th USENIX Symposium on Networked Systems Design & Implementation*, Cambridge, MA, USA, April 2007, pp. 229–242.

[31] 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," in *Proc. of the 5th IEEE Int. Conf. on Autonomic Computing (ICAC'08)*, Chicago, IL, USA, Jun. 2008.

[32] I. Aib and R. Boutaba, "On Leveraging Policy-Based Management for Maximizing Business Profit," *IEEE Trans. on Network and Service Management*, vol. 4, no. 3, pp. 25–39, 2007.