Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds

Ripal Nathuji and Aman Kansal
Microsoft Research
Redmond, WA 98052
{ripaln, kansal}@microsoft.com

Alireza Ghaffarkhah
University of New Mexico
Albuquerque, NM 87131
alinem@ece.unm.edu

Abstract

Cloud computing offers users the ability to access large pools of computational and storage resources on demand. Multiple commercial clouds already allow businesses to replace, or supplement, privately owned IT assets, alleviating them from the burden of managing and maintaining these facilities. However, there are issues that must be addressed before this vision of utility computing can be fully realized. In existing systems, customers are charged based upon the amount of resources used or reserved, but no guarantees are made regarding the application level performance or quality-of-service (QoS) that the given resources will provide. As cloud providers continue to utilize virtualization technologies in their systems, this can become problematic. In particular, the consolidation of multiple customer applications onto multicore servers introduces performance interference between collocated workloads, significantly impacting application QoS. To address this challenge, we advocate that the cloud should transparently provision additional resources as necessary to achieve the performance that customers would have realized if they were running in isolation. Accordingly, we have developed Q-Clouds, a QoS-aware control framework that tunes resource allocations to mitigate performance interference effects. Q-Clouds uses online feedback to build a multi-input multi-output (MIMO) model that captures performance interference interactions, and uses it to perform closed loop resource management. In addition, we utilize this functionality to allow applications to specify multiple levels of QoS as application Q-states. For such applications, Q-Clouds dynamically provisions underutilized resources to enable elevated QoS levels, thereby improving system efficiency. Experimental evaluations of our solution using benchmark applications illustrate the benefits: performance interference is mitigated completely when feasible, and system utilization is improved by up to 35% using Q-states.

Categories and Subject Descriptors C.0 [*General*]: System architectures; C.4 [*Performance of Systems*]: Modeling techniques; D.4.8 [*Operating Systems*]: Performance—Modeling and prediction; K.6.4 [*Management of Computing and Information Systems*]: System Management

General Terms Design, Management, Performance

Keywords Virtualization, Cloud computing, Resource management

1. Introduction

Cloud computing is rapidly gaining prominence, as evidenced by the deployment and growth of commercial cloud platforms [2, 9, 18]. The existence of these services enables businesses to replace, or dynamically supplement, their own IT infrastructures with large pools of computational and storage resources that are available on demand. While these consolidated infrastructures provide significant scale and efficiency advantages, there are still issues which prevent their widespread adoption. In particular, an important problem that remains to be effectively addressed is how to manage the quality-of-service (QoS) experienced by customers that share cloud resources.

Today, cloud providers charge customers based upon usage or reservation of datacenter resources (CPU hours, storage capacity, network bandwidth, etc), and service level agreements (SLAs) are typically based on resource availability. For example, a cloud provider may make guarantees in terms of system uptime and I/O request reliability. However, current systems do not make any application layer quality-of-service (QoS) guarantees. In environments where a given resource usage directly translates to application QoS, this is reasonable since customers deterministically receive levels of QoS based upon their purchased resource capacities. In shared cloud infrastructures, though, this is often not the case. Cloud platforms routinely employ virtualization [3, 32, 35] to encapsulate workloads in virtual machines (VMs) and consolidate them on multicore servers. Virtualization helps enable cohosting of independent workloads by providing fault isolation, thereby preventing failures in one application from propagating to others. However, virtualization does not guarantee performance isolation between VMs [16]. The resultant performance interference between consolidated workloads obfuscates the relationship between resource allocations and application QoS.

For customers, performance interference implies that paying for a quantity of resources does not equate to a desired level of QoS. For example, an application using one core of a multicore processor may experience significantly reduced performance when another application simultaneously runs on an adjacent core, due to an increased miss rate in the last level cache (LLC) [8, 13, 39]. One approach to deal with this indeterminism is to improve virtualization technologies, through better resource partitioning in both hardware and software, and remove performance interference altogether. A benefit of this approach is that the cloud can continue to be agnostic of application QoS and maintain simple resource capacity based provisioning and billing. However, perfect resource partitioning can be difficult and costly to implement, and even if accomplished, may result in inefficient use of resources [36].

To overcome the challenges imposed by performance interference effects, we advocate an alternative approach: QoS-aware clouds that actively compensate for performance interference using closed loop resource management. We present *Q-Clouds*, a QoS-aware control theoretic management framework for multicore cloud servers. Q-Cloud servers manage interference among consolidated VMs by dynamically adapting resource allocations to applications based upon workload SLAs. The SLAs are defined in terms of application specific performance metrics, and online adaptation is achieved through simple semantic-less [14] feedback signals. *Q-Clouds ensures that the performance experienced by applications is the same as they would have achieved if there was no performance interference.*

The Q-Clouds system makes multiple contributions. First, Q-Clouds employs application feedback to build multi-input multi-output (MIMO) models that capture interference relationships between applications. An advantage of the approach is that the system does not need to determine the underlying sources of interference. The MIMO model is used in a closed loop controller to tune resource allocations and achieve specified performance levels for each VM. Our second contribution builds on the Q-Clouds performance driven resource management to increase resource utilizations. In particular, applications may specify multiple QoS levels denoted as Q-states. Here, the lowest Q-state is the minimal performance that an application requires, but higher Q-states may be defined by the customer when they are willing to pay for higher levels of QoS. Q-Clouds uses this information to provision underutilized resources to applications when possible, while still guaranteeing that accompanying performance interference effects do not cause collocated applications to experience reduced QoS. We evaluate Q-Clouds on Intel Nehalem based multicore hardware with the Hyper-V virtualization platform. Our experiments with benchmark workloads, which exhibit performance degradations of up to 31% due to interference, show that Q-Clouds is able to completely mitigate the interference effects when sufficient resources are available. Additionally, system utilization is improved by up to 35% using application Q-states.

2. The Need for QoS-Aware Clouds

Managing application performance and QoS remains a key challenge for cloud infrastructures. The fundamental problem that arises in these environments is that *application performance can change due to the existence of other virtual machines* on a shared server [16, 31]. Moreover, the presence of other applications cannot be controlled by the customer. As a result, cloud promises on resource availability and capacity do not guarantee application performance. In this section we highlight the performance interference problem and discuss why adopting resource partitioning approaches to mitigate such effects are undesirable for cloud scenarios. We argue that interference should instead be addressed proactively with performance SLAs, as proposed in Q-Clouds.

2.1 Performance Interference with Consolidated VMs

If each customer application were allowed to run on dedicated compute, network, and storage resources, there would be no interference. In that situation, ignoring heterogeneous hardware [22], resource level SLAs in effect provide application QoS guarantees. However, the key advantages of cloud platforms come from efficient resource sharing and scaling. The nature of resource sharing is governed by two technology trends: virtualization and multicore processing. Virtualization [3, 32, 35] is employed for fault isolation and improved manageability, while multicore designs allow processors to continue leveraging Moore's Law for performance benefits. However, since each hosted workload may not be parallelized or scale-up to make use of all cores within a server, it becomes imperative to consolidate multiple workloads for efficient hardware utilization. The workloads, encapsulated in VMs, are then allocated "virtual" processors (VPs) that are backed by individual cores, or fractions of cores.

While virtualization helps to isolate VMs with respect to faults and security [20], it does not provide perfect performance isolation. For example, consolidating VMs onto a multicore package that incorporates a shared last level cache (LLC) creates an opportunity for the VMs to interfere with each other. We experimentally illustrate this problem using data collected on a quad-core Nehalem processor with an 8MB LLC. To localize the effects of cache interference, we first disable hardware prefetching mechanisms. We then measure the performance, in terms of execution time, of a simple synthetic CPU bound microbenchmark running in a VM. The microbenchmark is written such that the application iterates over a specified working set size multiple times until it has accessed a constant amount of data (2GB) which is significantly larger than the working set size. Therefore, as the working set size increases, the number of total iterations decreases to maintain a constant amount of work. Figure 1 provides the data obtained from this experiment.

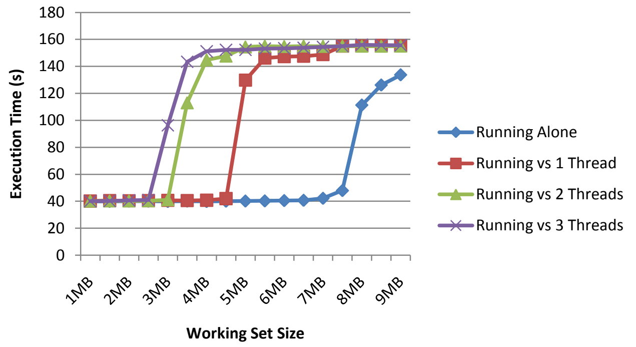


Figure 1. Performance impact of cache interference on consolidated VMs.

As illustrated in the figure, when a VM is running by itself, its performance degrades once the working set becomes larger than the LLC since it is effectively memory bound. We compare this performance to the case where the single VM is consolidated with a second VM running synthetic memory bound threads. The number of collocated threads varies from one to three, giving a total of up to four active threads, equal to the number of physical cores. Depending on the working set size, the performance of our first VM with a resource guarantee of one core, is significantly impacted, by up to 380%, due to LLC sharing.

2.2 Resource Partitioning to Mitigate Interference

For cloud systems, performance variations for the same resource usage, as seen in Figure 1, are not acceptable. One approach to handle performance interference is to better enforce resource partitioning in the shared system. For instance, the LLC can be effectively partitioned among VMs using page coloring [37] to avoid interference due to LLC sharing. Figure 2 illustrates the effects of page coloring in our previous experiment. Here, each of the two VMs are guaranteed their own 4MB partition of the cache. As expected, the existence of additional threads in the other VM no longer causes performance interference.

Based upon the page coloring results in Figure 2, resource partitioning may seem like a promising direction to solve the performance interference problem. However, adopting this approach is not plausible for two reasons. First, there is a cost in terms of system complexity when implementing partitioning mechanisms such as page coloring. Moreover, there

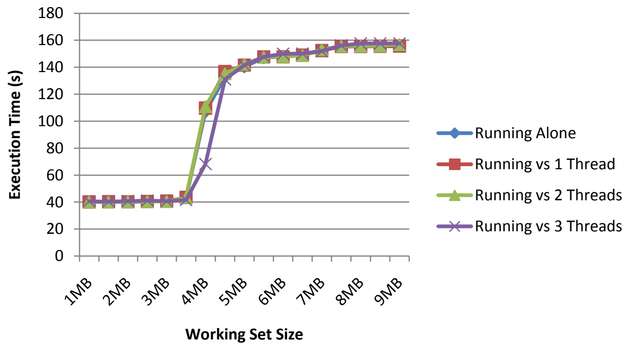


Figure 2. Mitigating LLC interference with VM page coloring.

are several dimensions of performance interference such as shared I/O and memory bandwidths [16], placing additional requirements on both hardware and software for effective partitioning across all of them. Secondly, even if the complexity of partitioning mechanisms is acceptable, it is likely that partitioning would lead to inefficient use of resources. For example, without dynamic page recoloring, in our example we would have allocated all of our LLC to the two VMs. This could prevent us from consolidating additional VMs, causing inefficient processor usage. Even with hardware support for cache partitioning, there would be inefficiencies in cache utilization due to the fact that lines rarely used by one core could not be used by another due to the strict partitioning [36]. Finally, in some cases, technologies like HyperThreading [30] that are designed to increase resource efficiency naturally add interference in the system. Thus, perfect performance isolation with partitioning may neither be practical due to complexity nor efficient in terms of resource utilization. Therefore, we must employ an alternative approach that ensures applications experience the QoS expected from purchased cloud resources.

2.3 Managing Clouds with Performance SLAs

When performance interference effects are present, the relationship between the amount of resource guaranteed and the application QoS achieved is broken. As a result, there are various scenarios that can be considered in terms of the metric used to charge the customer and the metric that is guaranteed. Figure 3 lays out four possibilities, based on whether the cloud *charges by* resources or application performance and whether the cloud *guarantees* resource or application performance. We discuss the key aspects of each of these below.

Scenario 1: Our first scenario is agnostic of application performance. Guarantees and prices are based solely on resource usage. This approach is simple to implement, and widely adopted in current cloud implementations. However, due to performance interference, this method is not sufficient for customers since the existence of other workloads, outside of their control, can cause significant performance vari-

		Guarantee metric	
		Resource capacity	Application performance
Pricing metric	Resource capacity	Scenario 1	Scenario 3
	Application Performance	Scenario 2	Scenario 4

Figure 3. Pricing and guarantee scenarios for clouds.

ations. Indeed, with this scenario the cloud provider benefits if interference prone workloads get placed together as then the application performance is minimized and the customer is forced to pay for more resources to meet performance SLAs.

Scenario 2: Following from scenario 1, in the second case the cloud still guarantees resource capacities but the price charged to the customer is based on the actual performance that is experienced. Compared to the first scenario, here if the customer experiences reduced performance because of interference, their payment to the cloud provider is adjusted accordingly. This case is still detrimental to the customer since there is no guarantee of performance. Moreover, in this case the cloud is also impacted negatively since the revenue generated by contracting resources is not deterministic.

Scenario 3: In this scenario, the cloud guarantees that the performance SLAs of applications are met. The customer is charged, however, based upon the amount of resources required to meet the SLA. This variation on scenario 1 therefore provides the customer with the benefit of performance guarantees, but still has the drawback that the customer is inevitably responsible for the costs of additional resources that are required to meet the performance SLA when there is interference amongst consolidated VMs.

Scenario 4: Our final case combines the benefits of scenarios 2 and 3. The cloud manages applications in a manner that performance guarantees are observed. Moreover, the customer is charged in terms of that level of performance, even if the cloud must tune resource allocations due to performance interference. The interesting issue, then, is how the charge for a level of performance is determined. Recalling that resource usage dictates application performance in the absence of interference, pricing can be based upon the resources required to meet a level of QoS when an application is running in an isolated manner. Then, if interference does occur, the cloud tunes resource allocations so that the desired performance is still achieved without affecting the charge to the customer. This provides the best experience for the customer, and the cloud is motivated to optimize resource usage and minimize interference, leading to the most efficient configurations. It is clear, then, that this is the best alternative amongst our four scenarios. Realizing this case, however, requires integrating QoS-aware resource management into cloud computing systems.

2.4 Enabling QoS-Aware Clouds

The Q-Clouds system is designed to support QoS-aware management of cloud hosted applications under performance interference effects. In particular, we are concerned with platform-level interference issues. From a cloud management perspective, there are two implications of interference that motivate the functional requirements of Q-Clouds. First, when placing VMs to maximize resource utilization and efficiency, it is necessary to understand the resource usage and requirements of the encapsulated applications. This information allows packing algorithms to deploy VMs in a manner that requires the least number of physical servers to be kept online. Such placement decisions may result in SLA violations if workloads need more resources than expected due to interference with co-hosted VMs. In the absence of a reliable mechanism that can compute the extent of interference for any possible VM placement, a practical approach towards handling this issue is for the VM deployment agent to maintain a resource "head room" when deploying VMs. The head room allows a Q-Clouds enabled server to dynamically tune resource allocations. An interference predictor could be used to estimate headroom allocations [16], or an empirically determined value can be used. *The first requirement of Q-Clouds is then to repurpose unallocated resources from the head-room, when necessary due to interference, to maintain performance SLAs.*

A second implication of performance interference is that, due to large variations in the extent of interference, the assigned head rooms will by nature be conservative. Therefore it is likely that the overall system will go underutilized. From an efficiency perspective, this is undesirable. One way to address this is by allowing applications to use the excess resources from the left over head rooms. In a cloud environment, this is only beneficial if the excess resources do indeed increase the application performance for some applications, the respective customers are willing to pay for the additional performance, and if any resulting interference effects do not cause other applications to drop below their SLAs. We enable such resource allocations through application Q-states, where customers can specify additional discrete levels of QoS that would be desirable, and for each state, the additional cost they would be willing to incur. *The second requirement of Q-Clouds is to utilize application Q-states to increase system utilizations when there is remaining head room.*

3. Q-Clouds Management Architecture

3.1 System Overview

Figure 4 presents an architectural overview of the management components used in Q-clouds, which we discuss next. Methods for the consolidation policies used by the cloud scheduler are assumed available from prior work [4, 11, 15], and are summarized only briefly. The focus of this paper is on the platform level mechanisms required in the subsequent online control. However, we discuss all of these elements to provide a holistic view of Q-Clouds enabled datacenters.

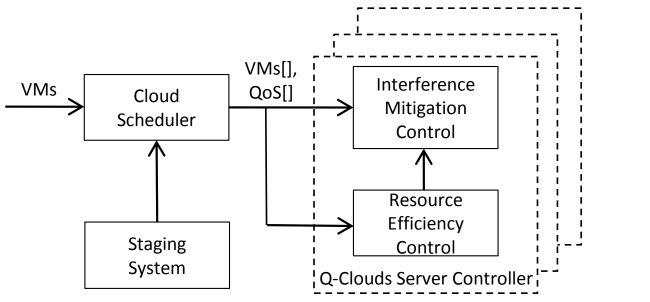

VMs
Cloud Scheduler
VMs[], QoS[]
Interference Mitigation Control
Resource Efficiency Control
Staging System
Q-Clouds Server Controller

Figure 4. VM Deployment and QoS management with Q-Clouds.

Cloud Scheduler: The *cloud scheduler* determines the placement of VMs submitted by customers on cloud servers. The cloud scheduler makes this decision based upon the resource requirements of workloads, as well as any other constraints that must be satisfied for security or reliability [29]. In order to determine resource requirements, VMs are first profiled on a *staging server* to determine the amount of resources needed to attain a desired level of QoS in an interference-free environment. The resource capacity determined here dictates what the VM owner will pay, regardless of whether additional resources must be provisioned at runtime due to performance interference. Once the resource requirement information is available, previously proposed solutions to the consolidation problem, such as those based on bin-packing [4, 11, 15] and those used in commercial clouds, may be employed. We assume any such solution is used, but with one variation: during VM placement, the cloud scheduler leaves a prescribed amount of unused resources on each server for use in subsequent online control. We refer to this unused capacity as "head-room" which, for example, may be determined empirically based on typical excesses used in interference mitigation. Upon completion, the cloud scheduler sends each Q-Clouds server a set of VMs along with their associated interference-free QoS and resource requirements.

Interference Mitigation Control: The head-room on each server is used to allocate additional resources to impacted VMs to bring their performance to the same level as what would have been observed without interference. In particular, we incorporate a closed loop *interference mitigation control* component that helps realize Scenario 4 from Section 2.3 by tuning resource allocations to achieve the desired performance for each VM. The controller operates on each server individually, where it employs a MIMO model to relate resource usage of all collocated VMs with their performance levels. We discuss the MIMO modeling and control approach in more detail in Section 3.2.

Resource Efficiency Control: Since the resource capacity used for interference mitigation is not known beforehand and the interference among VMs can vary, it is likely that the head-room allocated by the cloud scheduler ends up being conservative. The slack in resource can then be dynamically allocated to applications to achieve higher levels of QoS. In particular, we consider provisioning additional resources to applications that have multiple QoS levels defined via Q-states. We discuss Q-states in more detail in Section 3.3, but the goal of the *resource efficiency control* component is to determine when higher Q-states can be achieved, and to redefine target QoS points for interference mitigation control based upon the results of its optimization.

3.2 MIMO Modeling and Control

The control methods utilized by Q-Clouds rely upon the availability of a model that describes the relationship between resource allocations and the QoS experienced by VMs. A key capability of the system is that it learns this model online. A requirement to do so, however, is access to both the system inputs and the outputs. The inputs are readily available as control actuators to the underlying system. Access to the outputs, VM performance, requires active feedback from applications. To meet this need, the Q-Clouds platform relays QoS information from each VM on the server to the platform controller. In particular, we expose a paravirtualized interface that allows performance information to be communicated between virtual machines and an agent in the management partition (e.g. Root for the Hyper-V platform, or `Dom0` for Xen). A question, however, is what type of QoS data is required from each application. An artifact of cloud environments is that each application may have its own metric of performance. For example, compute applications may measure performance in millions of instructions per second (MIPS), while server workloads may use response time or request throughput. Our design does not use any semantic information regarding these performance metrics as the goal of Q-Clouds is simply to modify resource allocations so that the required level of QoS is met. This maps to a *tracking* control problem and it is sufficient for each application to provide its QoS information as a raw data value. This feedback is then used for two purposes: to build a system model and to drive online control of the platform.

In order to apply efficient resource allocation control, it is desirable for the system model to incorporate performance interference relationships between VMs that are consolidated onto a server. To meet this need, we adopt a multi-input, multi-output (MIMO) model approach which naturally captures performance interference interactions. Specifically, we consider a discrete-time MIMO model of the platform with p inputs and q outputs in order to design a model predictive control framework for Q-Cloud servers. The inputs $u_1[k], u_2[k], \ldots, u_p[k]$ of the model are defined to be the actuators used by the platform controller to manage resource allocations at time step k. For example, the system may utilize virtual processor (VP) capping mechanisms [23] on each VM to throttle the processing resources an application can use. The outputs $y_1[k], y_2[k], \ldots, y_q[k]$ of the model are the predicted QoS values at time step k. Let us denote by $\mathbf{u}[k] = \begin{bmatrix} u_1[k] & u_2[k] & \cdots & u_p[k] \end{bmatrix}^T$ and $\mathbf{y}[k] = \begin{bmatrix} y_1[k] & y_2[k] & \cdots & y_q[k] \end{bmatrix}^T$ the stacked vectors of the model inputs and outputs respectively. The general MIMO model of the platform is then given by a set of nonlinear difference equations as shown in Equation 1, where $\Phi()$ determines the outputs at time step k based upon previous outputs as well as current and previous inputs. The impact of previous outputs on the values at the current time step is determined by the parameter n and referred to as the *order of the system*. Similarly, the value m determines to what extent previous values of the input continue to impact the output at time step k. When n or m are nonzero, the current output depends on the history of prior inputs and outputs.

$$\mathbf{y}[k] = \Phi\big(\mathbf{y}[k-1], \cdots, \mathbf{y}[k-n], \mathbf{u}[k], \cdots, \mathbf{u}[k-m]\big) \quad (1)$$

In general, modeling of nonlinear dynamical system as depicted by Equation 1 is quite challenging. First, most nonlinear system identification techniques require significant computational complexity resulting in real-time implementation issues. Also, the large amount of learning data required for such modeling often makes the system identification of nonlinear dynamical systems impractical. However, in most applications, there exist simplified models that can capture the input-output interactions with a small amount of uncertainty. For example, static models are typically used for computing platforms experiencing very fast dynamics. In this case the platform model can be specified as $\mathbf{y}[k] = \Phi\big(\mathbf{u}[k]\big)$ with $n = m = 0$. We observed that in most of our experiments, a static approximation of the system model given by $\mathbf{y}[k] = \mathbf{A}\mathbf{u}[k] + \mathbf{b}$ is precise enough when the range of inputs under control is small, as is the case for interference mitigation control. Adopting this approximation enables the use of fast learning algorithms such as Least Mean Squares (LMS) or Recursive Least Squares (RLS) for finding the model parameters $\mathbf{A}$ and $\mathbf{b}$. Thus, for interference mitigation control, the model can be learned at run time as VMs get placed, allowing the controller to rapidly tune resource allocations as interference effects are observed.

It is important to note that the most accurate model in general can depend upon the workloads that are being considered, and the level of accuracy required may depend upon the type of control being applied. For example, if the control being applied requires capturing system dynamics and nonlinearities, neural networks might be applicable to better estimate the relationship in Equation 1 [24]. For our purposes, we evaluate and compare multiple cases in Section 5.1 to study these tradeoffs in our experiments. Once an appropriate MIMO model has been found, at any step we find optimal control inputs by solving a constrained optimization problem that leverages the predictive capabilities provided by the model. Taking the case where we model $\mathbf{y}[k] = \mathbf{A}\mathbf{u}[k] + \mathbf{b}$, Equation 2 provides an overview of the optimization based control, where $\mathbf{u}^*$ represents the optimal set of inputs based upon $\Lambda\big(\mathbf{y}\big)$ and $\Psi\big(\mathbf{y}\big)$. As denoted by $\text{argmin}_{\mathbf{u}}$ in the equation, we attempt to find among the possible input vectors $\mathbf{u}$, the one which minimizes $\Lambda\big(\mathbf{y}\big)$. For example, $\Lambda\big(\mathbf{y}\big)$ can quantify the deviation from the SLA outputs prescribed to the controller. The goal of the optimization is to find the minimum of this function where $\mathbf{u}$ is in the space of allowable inputs $\mathcal{U}$, and constraints on the output represented by $\Psi\big(\mathbf{y}\big) \leq 0$ are met. For example, where we want to meet some SLA outputs $\mathbf{y}_{SLA}$, $\Psi\big(\mathbf{y}\big) = \mathbf{y}_{SLA} - \mathbf{y}$. In this manner, $\Lambda\big(\mathbf{y}\big)$ and $\Psi\big(\mathbf{y}\big)$ are chosen based upon the desired QoS metrics for each of the consolidated VMs.

$$\begin{aligned} &\mathbf{u}^* = \text{argmin}_{\mathbf{u}}\ \Lambda\big(\mathbf{y}\big) = \text{argmin}_{\mathbf{u}}\ \Lambda\big(\mathbf{A}\mathbf{u} + \mathbf{b}\big) \\ &\text{s.t.} \\ &\quad \mathbf{u} \in \mathcal{U}, \\ &\quad \Psi\big(\mathbf{y}\big) = \Psi\big(\mathbf{A}\mathbf{u} + \mathbf{b}\big) \leq 0 \end{aligned} \quad (2)$$

As summarized above, our control approach is based upon a MIMO model that is learned and adapted online, and then used as a prediction tool for optimization based control. A benefit of this approach is that the model can also be used to determine whether provisioning additional resources to an application results in an improvement to QoS, and subsequently if such over provisioning may negatively impact other workloads. This ability can be used to determine when idle resources can be used to allow VMs to execute at higher QoS levels as described next.

3.3 Improving Cloud Efficiency with Q-states

For management purposes, we assume a minimum performance SLA is specified by customers for a VM deployed in the cloud. For example, a customer may request a level of QoS that requires half a processing core for a VM when it runs in isolation. Q-Clouds must then ensure that the application achieves that QoS during consolidated execution, even if additional resources must be supplied because of performance interference. Let us denote this level of QoS as Q0. For some customers, it may be that provisioning resources beyond those required to achieve Q0 may have additional value that they are willing to pay for. For example, certain computations may be able to vary their level of accuracy, where a base level of accuracy is denoted as Q0, but the customer may be willing to pay for marginal improvements beyond this [14]. Q-clouds allows customers to define additional Q-states that convey this desire so that servers can dynamically allot supplemental resources to applications, increasing the overall utilization and revenue of the cloud.

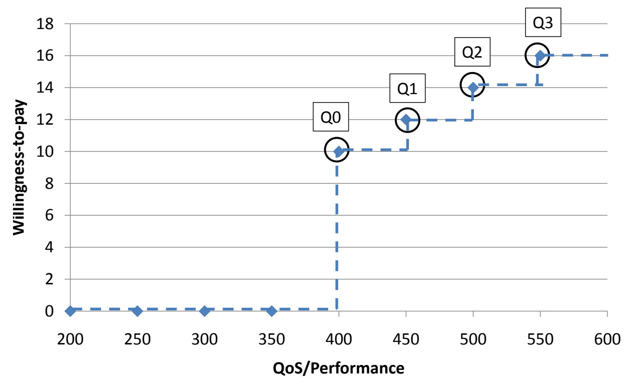


Figure 5. Example of application QoS and definition of Q-states and willingness-to-pay values.

The relationship between QoS and user benefit is often captured by continuous utility functions. Though continuous functions are convenient from a mathematical perspective, they are practically difficult for a user to define. Indeed, in the case where additional utility implies the willingness to pay extra for additional resources, it is likely that the mapping of utility to QoS may not even be continuous. Therefore, we support discrete Q-states, where we expect that it is much easier for customers to define a few levels of QoS, Qx, in addition to Q0 along with the amount of money that they would be willing to pay if the level of QoS is achieved. Figure 5 provides an example of this idea.

Given a set of Q-states for each VM, the platform controller can dynamically use up surplus head room on a server. In particular, the resource efficiency control component can perform an optimization to determine which, if any, VMs may be able to run at an elevated Q-state within the remaining resource capacity. At the same time, the controller must ensure that every VM still achieves its minimal QoS level of Q0. To do this the controller again makes use of our MIMO model. In this manner, Q-states allows future capabilities for customers to autonomously bid on and purchase additional resources only when it is beneficial to them (i.e. they get a performance benefit that pushes them to the next Q-state), while not disturbing other co-hosted applications. While in this paper we consider the use of Q-states to improve resource utilization, in the future we hope to investigate how they can help realize market based allocation of stranded datacenter resources [5].

4. Experimental Methodology

In this section we provide a brief overview of the experimental setup and implementation used to evaluate Q-Clouds enabled servers. We begin by defining the performance interference effects and workloads that we have chosen to consider. We then briefly describe our system implementation.

4.1 Performance Interference Effects and Workloads

As highlighted in previous work, performance interference can occur for various reasons including interference in CPU caches, memory bandwidth, and I/O paths [13, 16, 19, 31]. The Q-Clouds management framework is designed to handle any of these effects, with the availability of appropriate control actuators. Our current system includes the ability to cap the VP utilizations of guest VMs, thereby limiting accessibility to CPU resources [23]. Therefore, we limit our evaluation to interference effects that can be directly affected by varying CPU resources. In this context, there are three points of possible interference. First, multiple memory intensive applications can affect each other's performance due to interference in their respective memory bandwidth availability [19]. Second, as shown in Figure 1, applications can contend for resources in the last level cache (LLC). A final interference point is created by the use of hardware prefetching in processors. Here, the prefetchers employ a policy to determine when to prefetch cache lines, and scale back dynamically based upon bus utilization, etc. Figure 6 illustrates the inclusion of this affect when we enable prefetching in the experiments for Figure 1. Here, we see that even when the measured thread has a working set of 1MB and is not affected by sharing the LLC, its execution time doubles when it is running against three other threads due to reduced prefetching.

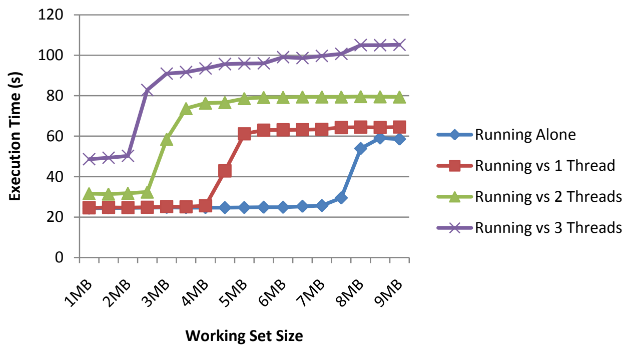


Figure 6. Performance impact of cache and prefetch hardware interference on consolidated VMs.

Based upon these multiple points of interference, and the availability of the VP capping actuator, in this paper, we focus on CPU bound workloads. In the future we plan to include additional control actuators so that we can better address I/O bound applications as well. For our evaluation, we use a set of benchmarks from the SPEC CPU2006 suite. Prior work has grouped these workloads according to their sensitivity to characteristics of the memory hierarchy (cache size, etc) [17]. As in Figure 6, our experiments utilize four applications running on the same quad-core processor. Therefore, we choose five of the SPEC CPU2006 benchmarks that are identified as being sensitive, and use all possible combinations of four as experimental workload mixes, shown in Table 1.

Table 1. SPEC CPU2006 workload mixes.

Benchmark	Workload Mix				
	1	2	3	4	5
436.cactusADM	X	X	X	X	
437.leslie3d	X	X	X		X
459.GemsFDTD	X	X		X	X
470.lbm	X		X	X	X
471.omnetpp		X	X	X	X

4.2 System Implementation

We implement and evaluate the Q-Clouds system using a dual socket enterprise server. Each socket is provisioned with a quad core Nehalem processor and 18GB of memory, for a total of eight cores and 36GB of memory on the platform. The Nehalem processor incorporates a three level cache hierarchy, where each core has its own L1 (32KB) and L2 (256KB) caches, and there is a large shared 8MB L3. Since this configuration restricts the interference effects we consider to each package, in our experiments we run four VMs that are consolidated onto one package, and leave the other idle. Finally, we deploy the Hyper-V virtualization software on the server, based upon the Windows 2008 R2 operating system [35].

Hyper-V utilizes a Root partition, similar to `Dom0` in Xen, that is able to perform privileged management operations on the platform. We implement a user space Root Agent that runs in the Root partition to drive the Q-Clouds resource management control and QoS monitoring. For resource monitoring, the agent can employ one of two methods. From our prior work, we have a simple paravirtualized interface that VM agents running in guests can use to convey QoS information over VMBus [23]. In addition, we have enabled the hypervisor to utilize performance counters in the hardware to monitor metrics on a per virtual processor (VP) basis. For example, we can monitor the number of instructions retired, cache accesses, cache misses, etc. The Root Agent can access this information from the hypervisor. In the context of this paper, since we focus on CPU bound applications whose performance QoS is based upon instructions executed, the Root Agent makes use of the hardware performance counter data to monitor the performance of each guest VM in terms of millions of instructions executed per second (MIPS).

As explained above, the Root Agent is able to monitor VM performance through the hypervisor. In addition, it has the ability to dynamically adjust CPU resource allocations provided to guest VMs by setting the associated VP utilization cap through the hypervisor. Based upon these inputs (VP caps) and outputs (guest QoS feedback), we implement our system controller in Matlab. In our lab setup, the Matlab component of the controller runs on a separate machine, and communicates with the RootAgent over the network. This setup is strictly due to ease of implementation, and there is no restriction that the controller be developed in this manner for our system. In our case, the Matlab component of the controller periodically queries the Root Agent for updated QoS feedback, and then invokes new VP caps through the Root Agent when necessary. As described in Section 3.2, the controller implements two pieces of functionality: 1) It uses feedback data to develop MIMO models of the consolidated VMs. 2) It uses the model to solve a constrained optimization problem in order to meet the SLAs of the applications.

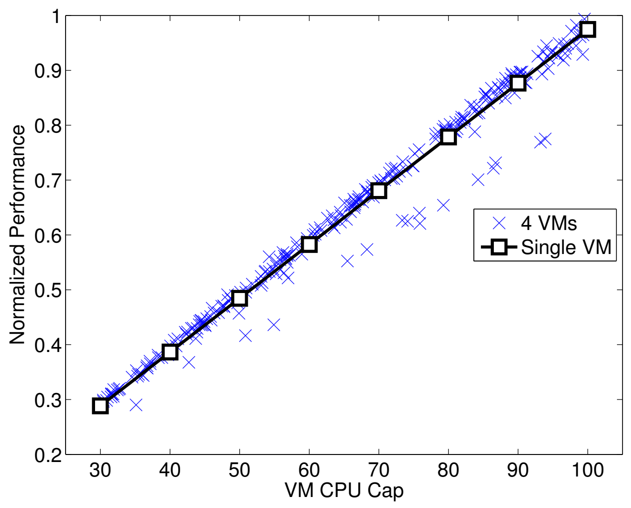
5. Evaluating Q-Clouds

5.1 Modeling VM Performance Interference

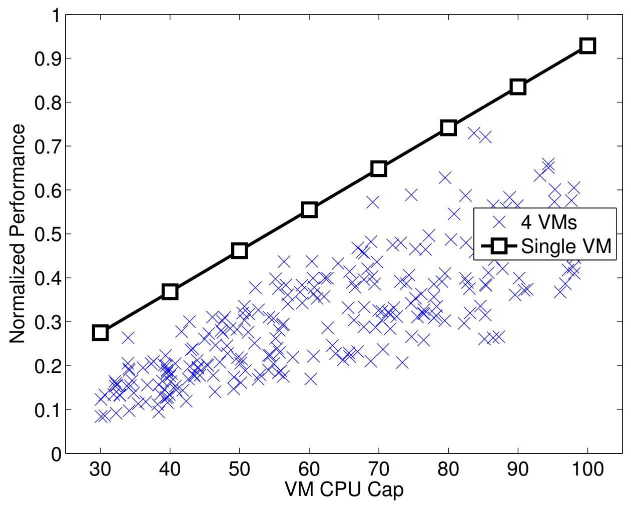
The feedback control implemented by Q-Cloud servers is predicated upon an available MIMO model that allows the controller to account for performance interference effects between consolidated applications. Figure 7 motivates the need for such a model. The figure illustrates the performance of our synthetic application from Figure 1 as a function of the CPU allocation (VP cap), comparing execution data when the application is running alone versus when it is consolidated with three other VMs running the same application for a total of four threads on a quad-core processor. Figure 7(a) illustrates the resulting performance when the working set size of the application is 1KB, and Figure 7(b) demonstrates how things change when the working set is made larger and interference begins to occur.

In both cases of Figure 7, we observe a linear relationship between performance and VP cap when there is a single VM running the application. We can compare this result against when we consolidate the remaining VMs and randomly vary the VP cap to all four. Both figures include the ensuing data points based upon the monitored performance of the first VM. As expected, when the working set is small, there is effectively no deviation from the original trend, and hence any performance model that captures the relationship between resource allocation and performance for the application by itself continues to hold under consolidation. However, in the case of the larger working set, we observe that the consolidated performance is strictly less than that achieved when the VM is running in isolation, and moreover, there is a wide variance. The latter can be attributed to the varying CPU allocations for the other three VMs, resulting in different amounts of interference. In general, the figure supports the fact that the system model used for control should be a function of all the workloads that a VM is consolidated with since performance interference interactions can vary.

We construct our system MIMO model using application performance feedback. In general, the interactions between applications can create nonlinear relationships. However, often you can model the nonlinear system as a linear system and it is accurate enough within a region of operation to perform control. The drawback, though, is that if you want to perform some optimized control over a larger region of the operating space (larger range of the controlled inputs), a lin-



(a) Application with 1KB Working Set Size



(b) Application with 3MB Working Set Size

Figure 7. Comparing the deviation from isolated performance as VMs are consolidated onto a multicore package.

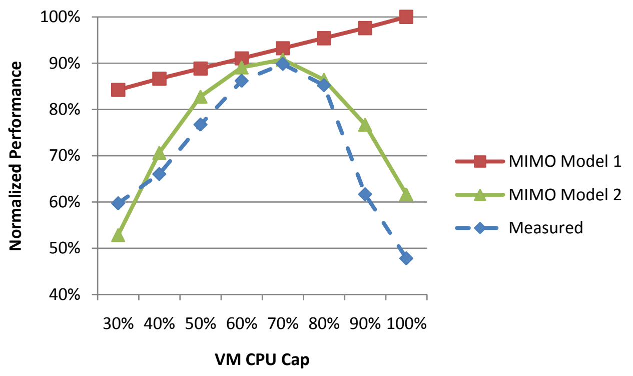


Figure 8. Comparison of MIMO modeling approaches.

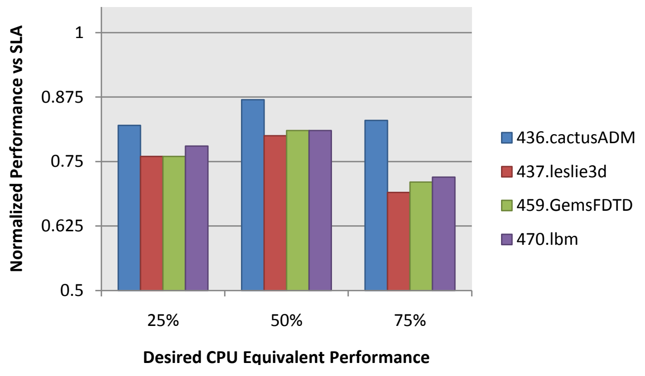
ear model may not allow you to find a good solution. Figure 8 illustrates this tradeoff with synthetic workloads that stress interference effects. The figure compares two modeling approaches. The first, MIMO model 1, uses the linear model described in Section 3.2 to capture the relationship between performance of VMs as a function of all VP caps. MIMO model 2 uses a more complex approach where the deviation from the linear trend between VP cap and performance when there is no performance interference is modeled with a second order polynomial. Here, we find the least squares solution for the coefficients of the polynomial to solve for the model.

In Figure 8, we plot the normalized performance of the system, where we have four VMs each running our synthetic workload with a 3MB working set. In each of the data points, the same VP cap is applied to all VMs. We can see from the measured data (dashed curve in the figure) that the operating point that maximizes performance is around 65%. If we wanted to predict this operating point with the MIMO models, the linear model would pick a poor solution since it predicts maximum performance at the operating point of 100%. Our second model, on the other hand, is able to capture the trends over the larger operating space much more accurately. The drawback of MIMO model 2, however, is that it is more complex, especially as we scale the number of system inputs. It also requires a model learning phase, possibly performed during system staging, where the VP caps of the VMs can be varied randomly across the entire operating region to accurately capture the interference relationships across a wider range of controlled inputs. On the other hand, MIMO model 1 can be adapted dynamically using a small history of observed data obtained during online control.

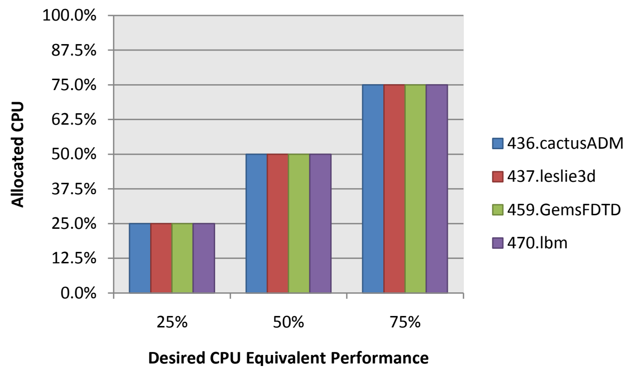
We observe that MIMO model 2 is of most benefit when we need to predict across the larger operating space. Indeed, our evaluations of the models show that in general, they both have similar error on average (approximately 13%) when you only need to predict within a small region. Therefore, we employ the models selectively depending on our needs. MIMO model 1 is used for interference mitigation control to achieve a specified set of SLAs (QoS points). This model is constructed completely online, and does not require learning during staging. We show in Section 5.2 that this light-weight approach allows Q-Cloud servers to control the system to appropriately meet SLAs. MIMO model 2 is then employed for resource efficiency control where we'd like to allocate remaining system resources to VMs to achieve higher Q-states. The resource efficiency control component can use the second MIMO model to quickly determine what set of Q-states are plausible given performance interference effects.

5.2 Meeting QoS Requirements with Q-Clouds

In evaluating the ability of Q-Cloud servers to meet QoS requirements under performance interference effects, we must assume the amount of head room that is available during consolidated execution. As previously mentioned, in this paper our focus is on the platform component of the Q-Clouds systems. Therefore, we assume different amounts of head room that may occur when VMs are deployed by the cloud scheduler. In particular, for each of the workload mixes in Table 1, we assume that all four VMs have SLAs that require 25%, 50%, or 75% of the CPU when there is no performance interference, resulting in head rooms of 75%, 50%, and 25% during consolidation. This gives us fifteen scenarios comprised of five workload mixes and three possible SLAs per mix. In our first set of experiments we deploy the four workloads in a mix, and define the desired SLAs based upon the performance experienced with the respective resource capacity when there is no interference. We compare against a default case where the system assumes that resources required to meet the SLA do not change as workloads are consolidated (i.e. QoS-unaware resource allocations).



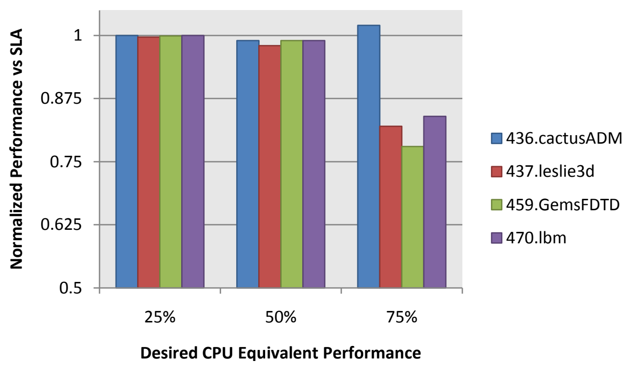
(a) Default Performance



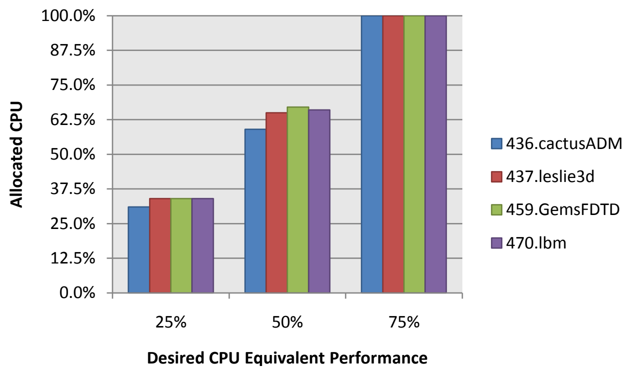
(b) Default Resource Allocation

Figure 9. Performance and resource allocations with a default QoS-unaware system for workload mix one.

Figure 9 provides results from our default case with the first workload mix. We observe that with QoS-unaware management, applications are consistently impacted by up to 30%. While some applications are affected more heavily than others, there is significant degradation across all of them. This clearly demonstrates the need for the type of online management provided by Q-Clouds. Taking this experiment and enabling interference mitigation control, we obtain the application performance and resource allocation data provided in Figure 10. Comparing Figures 9(a) and 10(a) we observe clearly that the Q-Clouds controller is able to use the MIMO model of the system to adapt resource allocations so that the desired performance is met, with the exception of the case where the desired performance is equivalent to 75% of the CPU. The reason for this can be observed in Figures 9(b) and 10(b). We see that when applications desire CPU equivalent performances of 25% and 50%, the Q-Clouds server allocates additional resources to the VMs in order to meet the desired QoS. However, in the case of 75%, the system allocates additional resources to the point that it is unable to provision additional capacity. Therefore, the inability of Q-Clouds to meet QoS requirements is due to the fact that there is not enough head room in the system for Q-Clouds to properly provision the VMs.



(a) Q-Clouds Performance



(b) Q-Clouds Resource Allocation

Figure 10. Performance and resource allocations with a Q-Clouds enabled system for workload mix one.

Figure 11 provides the performance data for the remainder of our workload mixes. The results are similar to our first mix, where Q-Clouds is able to utilize available resources from the consolidation step to provision additional resources to VMs so that QoS requirements can be met. Again, in the case where applications desire performance levels equivalent to 75% of a CPU, Q-Clouds is unable to meet the QoS for all workloads. Indeed, in some cases as in workload mix five, there is significant interference and none of the applica-

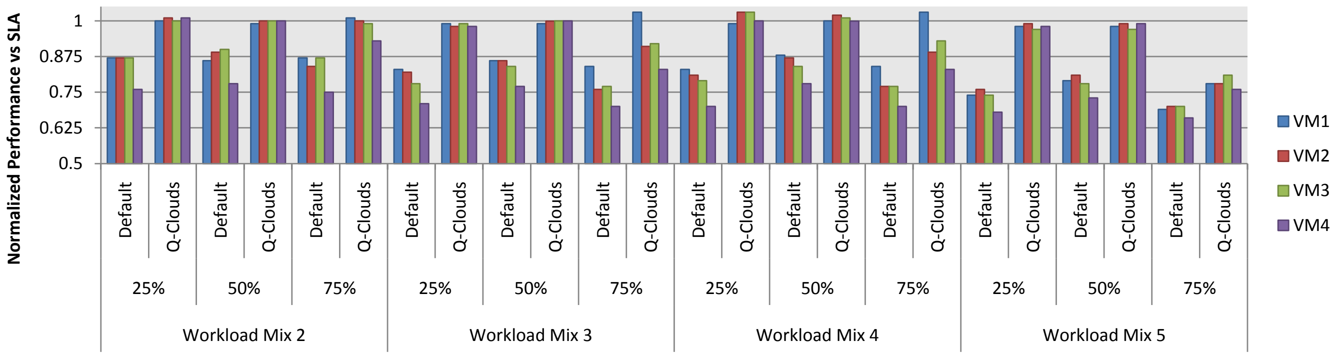


Figure 11. Performance comparisons of default and Q-Clouds for workload mixes two through five.

Table 2. Q-Clouds average CPU allocation after interference mitigation control.

Workload Mix	Default allocation of 25% CPU	Default allocation of 50% CPU	Default allocation of 75% CPU
1	33%	64%	100%
2	31%	58%	96%
3	32%	63%	100%
4	33%	63%	100%
5	36%	70%	100%

tions can meet their QoS. This is in spite of the fact that in this case, as shown in Table 2, Q-Clouds allocates the system fully to the applications.

Given the results in Figure 11 and Table 2, it is clear that provisioning VMs that require CPU equivalent performance of 75% does not leave ample head room in the system for Q-Cloud controllers to mitigate performance interference effects. On the other hand, allocations of 25% leave significant system resources unused, on average 67%. The intermediate case of 50% leaves between 30% and 42% of resources unused. As we show next, we can dynamically reprovision these resources to achieve higher QoS levels as defined by applications, thereby increasing resource efficiency and revenue for the cloud while providing better performance for customers.

5.3 Improving Cloud Utilization with Q-states

Building on the results in Section 5.2, we evaluate the inclusion of Q-states as described in Section 3.3. For our evaluation, we again take our five workload mixes, but assume in each case a head room of 50% where each VM requires performance equivalent to 50% of a CPU when there is no performance interference. By our definition of Q-states, these QoS points are Q0 states. We define three addition Q-states, Q1, Q2, and Q3, where for each VM these are defined to be CPU equivalent performance of 60%, 70%, and 80% respectively. We integrate MIMO model 2 in a global optimization step, where the Q-Cloud resource efficiency controller determines the set of Q-states that can be achieved. The idea is that it can use the MIMO model as a predictor to determine plausible operating points. It then defines the appropriate target performance points to the interference mitigation controller which from then on uses a dynamically adapted linear MIMO model 1 to steer the system towards the specified operating point.

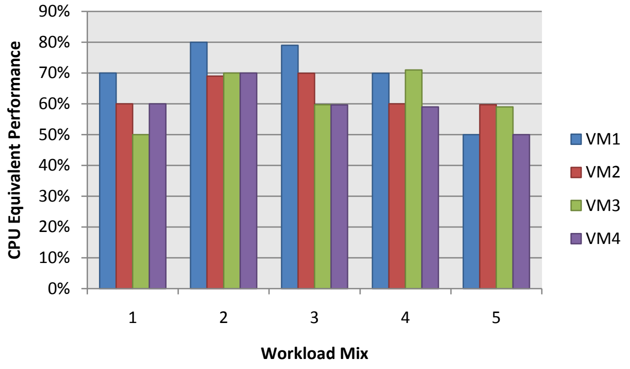


Figure 12. QoS provided to applications with Q-states: Some VMs experience higher Q-states than Q0 (CPU equivalent performance of 50%).

Figure 12 provides the QoS levels provided by the Q-Cloud server. We observe that the system is indeed able to identify appropriate Q-states and achieve them. Depending on the workload mix, the system is able to provide various levels of QoS improvement. For example, in workload mix five, there is significant interference between applications, so two of them are limited to Q0 while the other two receive Q1. On the other hand, in workload mix two there is significant flexibility, and the system is able to provision VM1 with Q3 and the others with Q2. In general, the system is able to leverage the predictive capabilities of the more complex MIMO model 2 to find an operating point that is reachable without overly straining the system. Though we do not consider the ensuing revenue generated by running at higher Q-states, the Q-Clouds controller can allow customers to bid with willingness to pay values and determine the optimal allocation in terms of revenue. Here we validate the benefits of the mechanism, and plan to further investigate its ability to enable dynamic market driven resource allocation for clouds [5].

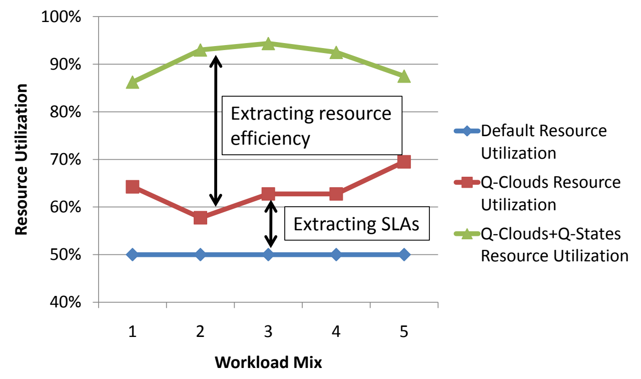


Figure 13. Resource utilization comparisons between default systems, Q-Clouds servers, and Q-Clouds with Q-state enabled resource efficiency optimization.

As a final result, Figure 13 summarizes the system-level benefits of the Q-Clouds system. We observe the first step in resource utilization where the Q-Clouds platform controller increases resource allocations in order to meet the minimal Q0 level of QoS. Then, based upon application defined Q-states, the system tries to provision remaining resources to meet higher QoS levels, while still accounting for performance interference effects so that all other applications are still able to meet their SLA. We observe that the use of interference mitigation control to meet Q0 levels of performance uses up additional resources by up to 20%. In addition, employing resource efficiency control increases system utilization by another 35%. Overall, the average Q-Cloud server utilization is 91%. This illustrates how the combination of Q-states and closed loop control can significantly improve resource utilizations and efficiencies even in the presence of performance interference effects.

6. Related Work

Performance interference effects from shared resources can significantly impact the performance of co-located applications. Hardware extensions to enable improved resource management of shared processor resources such as caches have been considered extensively in previous work. For example, hardware techniques have been designed that dynamically partition cache resources based upon utility metrics [27], or integrate novel insertion policies to pseudo-partition caches [36]. Similarly, changes to the memory subsystem have been developed that can help manage interference effects in memory bandwidth [19]. Mechanisms to enable improved monitoring of shared cache structures to detect and quantify interference have been proposed as well [38]. More sophisticated techniques build on these monitoring schemes to enable priority driven management of resources in hardware for improved QoS support [13]. Similar to operating system scheduling techniques for performance isolation [8, 39], we approach the problem of performance interference based upon existing hardware platform capabilities. Moreover, with Q-Clouds, our goal is to address multiple sources of performance interference by dynamically provisioning additional resources when necessary to meet QoS objectives of virtualized applications, without having to fully understand the details of where and how interference is occurring. However, the availability of additional hardware support for monitoring and actuation can help improve both modeling and control of the overall system, and would be invaluable for instantiations of Q-Clouds on future generations of hardware which incorporate these types of features.

A key benefit of virtualization is the flexibility it provides to easily manage applications. The disassociation between virtual and physical resources allows the management layer to transparently adapt resource allocations [21, 25, 26]. In addition, the integration of live migration technologies [6] introduces the ability to adaptively allocate VMs across distributed servers. This can be used to pack VMs in a manner that minimizes unused resources, where the allocator may even forecast future resource needs [4, 15]. More sophisticated schemes may consider the overheads and sequences of migrations necessary to achieve a desired cluster allocation [11]. However, awareness of performance interference effects between VMs is limited in these approaches. For example, prior attempts to intelligently place HPC workloads to avoid cache contention have been evaluated [31]. Approaches to predict performance interference [13, 16] can be used in conjunction with these solutions to prescribe a practical amount of excess resources on platforms. Once workloads have placed onto servers, Q-Clouds performs feedback driven control to provision resources using a MIMO model that captures interference effects from multiple points.

Feedback control methods have been applied for resource management in various environments. For example, controllers targeted for the specific case of web server applications have been developed [1]. Other instances have considered the use of control theoretic methods to concurrently manage application performance and power consumption [23, 34]. In the case of virtualized systems, prior work has evaluated the use of feedback control for resource allocation between tiers of multi-tier applications [25] to meet SLAs, where the approach can be extended to include multiple resources beyond just the CPU using MIMO models [26]. Recognizing the proliferation of controllers across the system, methods to coordinate distributed controllers have also been investigated [28]. The control techniques used in our Q-Clouds system compliment these methods, while specifically addressing the issue of resource management under performance interference effects for cloud environments.

7. Conclusions and Future Work

Cloud computing is a rapidly emerging paradigm with significant promise to transform computing systems. Indeed, multiple commercial solutions are beginning to provide cloud services and resources [2, 9, 18]. Today, customers are charged based upon resource usage or reservation. However, the performance that an application will obtain from a given amount of resource can vary. A significant source of variation is from performance interference effects between virtualized applications that are consolidated onto multicore servers. It is clear that for clouds to gain significant traction, customers must have some guarantees that a quantity of resource will provide a deterministic level of performance. In this paper we present our Q-Clouds system that is designed to provide assurances that the performance experienced by applications is independent of whether it is consolidated with other workloads. Contributions of our system include the development and use of a MIMO model that captures interference effects to drive a closed loop resource management controller. The controller utilizes feedback from applications to meet specified performance levels for each VM. We build on this capability by introducing the notion of Q-states, where applications can specify additional levels of QoS beyond a minimum that they are willing to pay for. Q-Cloud servers can make use of these states to provision idle resources dynamically, thereby improving cloud efficiency and utilizations.

In this paper we have implemented and evaluated Q-Cloud servers based upon interference effects between CPU bound applications. Though this work provides a solid foundation, there are multiple challenging problems that are not addressed here which we hope to pursue in the future. First, we plan to investigate how Q-Clouds can be extended to better address interference in the I/O paths. As part of this, we will need to extend the platform virtualization infrastructure with appropriate control actuators. This will require accounting for evolving hardware mechanisms which improve platform level I/O virtualization [7], as well as how we might leverage exposed control and monitoring mechanisms from existing technologies that help enable performance isolation for storage resources that are shared amongst multiple virtualized hosts [10, 33]. In evaluating these enhancements, we would also exercise Q-Clouds with a more diverse set of workloads.

A second direction that was not discussed in this paper concerns issues around applications with phased behaviors. Depending upon the relative time constants between phased behavior and the windows across which SLAs are defined and enforced, it may be necessary to manage a system in a phase-aware manner. In order to address this need, we can consider if and how prior work on phase monitoring and prediction can be applied to our control and modeling approaches [12]. A final direction of future work includes investigating the integration of performance interference aware control for dynamic workload placement using live migration techniques. Here we hope to better understand how different application mixes affect the overall benefits of our solution, and if workload heterogeneity can be exploited to make better use of cloud hardware under QoS constraints.

References

[1] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), 2002.

[2] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2`.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[4] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2007.

[5] A. Byde, M. Salle, and C. Bartolini. Market-based resource allocation for utility data centers. Technical Report HPL-2003-188, HP Laboratories, September 2003.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[7] Y. Dong, Z. Yu, and G. Rose. Sr-iov networking in xen: Architecture, design and implementation. In *Proceedings of the First Workshop on I/O Virtualization (WIOV)*, December 2008.

[8] A. Fedorova, M. Seltzer, and M. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2007.

[9] Google App Engine. `http://code.google.com/appengine`.

[10] A. Gulati, I. Ahmad, and C. A. Waldspurger. Parda: Proportional allocation of resources for distributed storage access. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, February 2009.

[11] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, 2009.

[12] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2006.

[13] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. Vm3: Measuring, modeling and managing vm shared resources. *Journal of Computer Networks*, 53(17), 2009.

[14] A. Kansal, J. Liu, A. Singh, R. Nathuji, and T. Abdelzaher. Semantic-less coordination of power management and application performance. In *Workshop on Power Aware Computing and Systems (HotPower)*, October 2009.

[15] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

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

[17] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2008.

[18] Microsoft Azure Services Platform. `http://www.microsoft.com/azure`.

[19] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *Proceedings of the 16th USENIX Security Symposium*, 2007.

[20] D. G. Murray, G. Milos, and S. Hand. Improving xen security through disaggregation. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, 2008.

[21] R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, October 2007.

[22] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[23] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *Proceedings of the Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, April 2009.

[24] M. Norgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems*. Springer, April 2003.

[25] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Proceedings of the EuroSys Conference*, 2007.

[26] P. Padala, K.-Y. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the EuroSys Conference*, 2009.

[27] M. Qureshi and Y. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2006.

[28] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: Coordinated multi-level power management for the data center. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[29] H. Raj, R. Nathuji, A. Singh, and P. England. Resource management for isolation enhanced cloud services. In *Proceedings of the Cloud Computing Security Workshop (CCSW)*, 2009.

[30] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1995.

[31] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of hpc applications. In *Proceedings of the International Conference on Supercomputing (ICS)*, 2008.

[32] VMware ESX. `http://www.vmware.com/products/esx`.

[33] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger. Argon: performance insulation for shared storage servers. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, February 2007.

[34] X. Wang and Y. Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *Proceedings of the 17th IEEE International Workshop on Quality of Service (IWQoS)*, Charleston, South Carolina, July 2009.

[35] Windows Server 2008 R2 Hyper-V. `http://www.microsoft.com/hyperv`.

[36] Y. Xie and G. H. Loh. Pipp: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2009.

[37] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the EuroSys Conference*, March 2009.

[38] L. Zhao, R. Iyer, R. Illikkal, J. Moses, S. Makineni, and D. Newell. Cachescouts: Fine-grain monitoring of shared caches in cmp platforms. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2007.

[39] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.