NINEPIN: Non-Invasive and Energy Efficient Performance Isolation in Virtualized Servers

Palden Lama and Xiaobo Zhou
Department of Computer Science
University of Colorado at Colorado Springs, CO 80918, USA
{plama, xzhou}@uccs.edu

***Abstract*—A virtualized data center faces important but challenging issue of performance isolation among heterogeneous customer applications. Performance interference resulting from the contention of shared resources among co-located virtual servers has significant impact on the dependability of application QoS. We propose and develop NINEPIN, a non-invasive and energy efficient performance isolation mechanism that mitigates performance interference among heterogeneous applications hosted in virtualized servers. It is capable of increasing data center utility. Its novel hierarchical control framework aligns performance isolation goals with the incentive to regulate the system towards optimal operating conditions. The framework combines machine learning based self-adaptive modeling of performance interference and energy consumption, utility optimization based performance targeting and a robust model predictive control based target tracking. We implement NINEPIN on a virtualized HP ProLiant blade server hosting SPEC CPU2006 and RUBiS benchmark applications. Experimental results demonstrate that NINEPIN outperforms a representative performance isolation approach, Q-Clouds, improving the overall system utility and reducing energy consumption.**

Keywords: Performance Isolation, Non-invasiveness, Virtualized Servers, Energy Efficiency, Robustness, Fuzzy MIMO Control

I. Introduction

A modern data center utilizes virtualization technology to consolidate multiple customer applications onto high density servers for improving server utilization and reducing energy consumption costs [5], [9], [23], [27]. It also aims to satisfy the Quality of Service (QoS) needs of hosted applications for increasing data center utility. However, QoS experienced by these applications may be significantly impacted by the performance interference between virtual machines (VMs) that are co-located in the underlying multi-core servers [11], [21]. It is mainly due to the contention of resources such as the last level cache, memory bandwidth, etc, which are shared by VMs residing on a multi-core processor. For instance, VMs running on adjacent CPU cores may experience significantly reduced performance due to an increased miss rate in the last level cache [3], [31]. Performance isolation is essential to dependable virtualized servers shared by various applications.

In this paper, we propose to design and develop a non-invasive and energy efficient performance isolation mechanism that increases the overall utility of a virtualized server system hosting heterogeneous customer applications. It is important but challenging to achieve performance isolation between Internet applications running on virtualized servers.

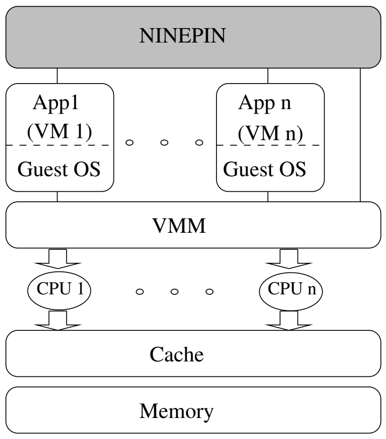


Fig. 1. NINEPIN: non-invasive performance isolation in virtualized servers.

There are invasive techniques based on hardware and software resource partitioning, which require instrumentation and modification of the guest operating system or the virtualization management layer to avoid performance interference between co-located VMs [2], [24], [29], [30]. However, resource partitioning can be difficult and costly to implement and even if accomplished may result in inefficient resource utilization indeed [28]. Due to portability and transparency needs, non-invasive performance isolation is desirable in the context of modern data centers provisioning cloud computing services, which host third-party customer applications and often use virtualization software from third-party vendors.

From a data center's economic perspective, performance isolation should be aligned with the incentive to maximize the overall system utility, which includes the service-level utility of customer applications and the utility of server energy consumption. A service-level utility function specifies the business value of providing various levels of service to the users of an application in terms of revenue or penalty [26]. The utility of energy consumption is determined by the electricity costs as well as the carbon footprint associated with it. However, existing performance isolation techniques are utility-agnostic.

A naive approach that disregards the economic perspective may achieve performance isolation by allocating additional resources to compensate the effect of performance interference among co-located VMs. Recently an important non-invasive performance isolation approach, Q-Clouds [21] was proposed to ensure that the performance experienced by applications is the same as they would have achieved if there was no performance interference. However, such an approach does not guarantee optimal operating conditions with respect to energy efficiency of underlying servers and the service-level utility of hosted applications.

Furthermore, there are practical issues in achieving robust performance isolation among heterogeneous applications co-located in the same physical server. These applications may have different performance metrics as well as workload dynamics. For instance the performance of compute intensive jobs is measured in terms of how fast a job is completed whereas interactive web applications with multi-tier architecture are concerned with end-to-end response time of requests. Furthermore an interactive web application shows dynamic workload variations at small time scales [13], [16], [25]. As a result, co-located VMs experience frequently changing performance interference effects and even resource saturation. Moreover, energy consumption characteristic of VMs may also depend on the workload intensity. Hence, it is very challenging to achieve performance isolation and energy efficiency at the same time in a heterogeneous application environment.

In this paper, we propose and develop NINEPIN, a non-invasive and energy efficient performance isolation mechanism that mitigates the performance interference between heterogeneous customer applications hosted in virtualized servers. As shown in Figure 1, NINEPIN interacts with the virtualization management layer of a multi-core server and the co-located VMs at the application layer. Its core is a novel two-level control structure. The first level performs a steady-state utility optimization that aims to maximize the overall system utility. It determines the economically optimal performance targets for each application and sends these targets to the second level, the model predictive controller. The controller regulates the system's dynamic behavior towards the optimal targets by adjusting the allocation of resources among co-located applications. The utility optimization and control are based on system models that capture the performance interference relationship between co-located applications and the total energy consumption of the underlying physical server for various resource allocations.

NINEPIN constructs fuzzy multiple-input multiple-output (MIMO) models for estimating the performance interference and energy usage in a virtualized server when different CPU usage limits are enforced on the co-located VMs. A key strength of fuzzy MIMO model is its ability to accurately represent the inherently non-linear relationship of performance and energy with CPU usage. NINEPIN applies subtractive clustering and artificial neural network based machine learning techniques to construct the performance interference and energy usage models. In order to achieve system robustness against dynamic workload variation and application heterogeneity, it adapts the models online by use of a fast learning algorithm, weighted Recursive Least Squares (wRLS), whenever a significant error in prediction of energy usage and performance is detected. Then, it re-computes the optimal performance targets using the updated performance interference and energy models. The model predictive controller uses a dynamic model that is derived by linearizing the fuzzy MIMO model at the current operating state.

We implement NINEPIN on a testbed of HP ProLiant BL460C G6 blade server hosting SPEC CPU2006 benchmark applications and an e-commerce benchmark application RUBiS. The testbed uses VMware virtual machines. Due to its non-intrusiveness, NINEPIN is applicable to any virtualization software given that the mechanisms to adjust VM resources are available. Experimental results demonstrate the effectiveness and energy efficiency of NINEPIN in achieving performance isolation among multiple heterogeneous customer applications. For performance comparison, we also implement the representative non-invasive performance isolation approach, Q-Clouds [21] at the same testbed. Q-Clouds uses a closed loop controller to compensate the effect of performance interference between co-located VMs by allocating additional resources. However, such an approach does not guarantee optimal operating conditions with respect to energy efficiency of underlying servers and the overall system utility. It also does not consider heterogeneous application support.

Compared to Q-Clouds, NINEPIN achieves better system utility and significantly reduces energy consumption. We have observed that the advantage of NINEPIN approach is even more significant in case of heterogeneous applications with dynamic workload variations. NINEPIN is able to re-compute and assure optimal performance targets in response to the dynamic environment in agile and robust manner.

To our knowledge, NINEPIN is the first non-invasive performance isolation mechanism that drives a virtualized server system towards optimal operating conditions with respect to both energy efficiency and service-level utility of hosted applications. The main contributions of NINEPIN are:

1) It provides effective performance isolation between co-located applications in virtualized servers while maximizing the overall system utility. It increases data center utility by aligning performance isolation goals with a data center's economic optimization objective.
2) It is energy efficient. It reduces the energy consumption of virtualized servers while trading off performance objectives in a flexible manner. The tradeoff between inherently conflicting objectives of energy efficiency and performance guarantee can be specified by a data center administrator.
3) It is robust against application heterogeneity and dynamic workload variations.
4) It provides desirable non-invasive performance isolation for a data center hosting third-party customer applications and using virtualization software from third-party vendors.

NINEPIN combines the strengths of machine learning based self-adaptive system modeling, utility based performance targeting and a model predictive control based target tracking. The two-level structure of NINEPIN integrates utility computing paradigm with control theoretical approach. Together with the strength of fuzzy logic, our novel hierarchical control framework achieves this complex integration while avoiding highly complex system modeling and computationally expensive control. Hence, NINEPIN is practical for real virtualized server systems. We demonstrate the merits of NINEPIN with implementation on a testbed of virtualized servers.

In the following, Section II discusses related work. Sections III-B through V present NINEPIN architecture and hierarchical control design. Section VI presents the testbed implementation. Section VII provides the experimental results and analysis. Section VIII concludes the paper.

II. Related Work

Performance isolation of customer applications in a virtualized data center is an important research topic. Despite several advantages such as security isolation, fault isolation, and environment isolation, prevalent virtualization techniques do not provide effective performance isolation between VMs [11], [21]. The behavior of one VM can affect the performance of another adversely due to the shared use of resources in the system. VMs running on the underlying multi-core servers of a virtualized data center mainly suffer from the performance interference caused by the contention of last level cache and memory bandwidth. The performance impact of shared resource contention in multi-core servers has been well studied in the studies [7], [11], [21].

Several research efforts have focused on hardware and software resource partitioning based techniques for performance isolation of applications running on a multi-core server. Hardware-based cache partitioning schemes are mainly involved with modification of cache replacement policies [30] with various partition granularity such as cache ways and cache blocks. On the other hand, software partitioning technique based on static and dynamic page coloring addresses cache contention between competing applications, without requiring any hardware level support [2], [24], [29]. Page coloring reserves a portion of the cache for each application, and allocates the physical memory such that the application's cache lines map only into the reserved portion. However, such approaches in virtualized servers require invasive instrumentation and modification of the guest operating system or the virtualization management layer.

Some prior studies investigated the design of cache-aware scheduling algorithms that achieves performance isolation among competing applications by minimizing resource contention [3], [10], [31]. For instance, Fedorova *et. al* designed a cache-aware scheduler that compensates threads that were hurt by cache contention by giving them extra CPU time [3]. Knauerhase *et. al* [10] proposed to reduce cache interference by spreading the cache intensive applications apart and co-scheduling them with non-intensive applications. A common drawback of cache-aware scheduling and resource partitioning based performance isolation mechanism is that they only focus on a single source of performance interference. However, in practice there are several dimensions of performance interference such as shared I/O and memory bandwidths [11].

Recently, Nathuji *al.* proposed an interesting non-invasive performance isolation approach for virtualized servers, Q-Clouds [21]. Q-Clouds builds MIMO models that capture interference relationships between co-located VMs and applies a closed loop controller to achieve specified performance levels for each VM. Due to its non-invasive nature, the approach does not need to determine the underlying sources of interference. However, it disregards the economic objective of a data center, which is defined by the service-level utility of customer applications. Furthermore, it does not consider energy efficiency and heterogeneous application support.

Energy consumption costs and the impact of carbon footprint on the environment have become critical issues for data centers today [4], [19]. There are recent studies that aim to guarantee fixed performance targets of data center applications while minimizing the power consumption [1], [8], [12], [15], [17], [18]. However, they do not consider the impact of performance interference between co-located VMs on the energy efficiency and the system utility.

III. NINEPIN Architecture and Design

A. Design Goals and Motivations

NINEPIN provides an attractive and practical non-invasive and energy efficient performance isolation mechanism for virtualized servers that host heterogeneous applications. It maximizes the overall system utility. The key design issues of NINEPIN are as follows:

1) *Non-invasiveness with utility optimization*: An intuitive approach of non-invasive performance isolation among co-located applications is to allocate additional resources to achieve the performance that customers would have realized if they were running in isolation. However, such approaches are inherently utility-agnostic. Integration of non-invasive performance isolation with utility optimization would require highly complex system modeling and computationally expensive control. NINEPIN addresses the challenge by using a novel hierarchical control framework.
2) *Energy efficiency*: A common technique to reducing server energy consumption is to dynamically transition the hardware components from high power states to low-power states. However, it is not applicable in case of virtualized servers since changing the power state of a processor will affect the performance of multiple VMs running different applications. NINEPIN achieves energy efficiency by controlling the CPU usage limits on each VM, based on an accurate energy model. It allows a data center administrator to flexibly trade-off energy consumption with the service-level utility of hosted applications.
3) *Robust performance isolation*: The robustness of performance isolation against application heterogeneity and dynamic workload variations requires a self-adaptive approach that responds to the changes in the performance

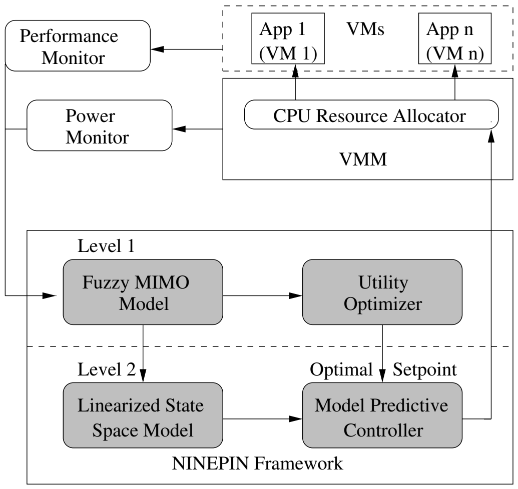


Fig. 2. NINEPIN System Architecture.

interference relationship and the energy consumption characteristic of the co-located VMs. NINEPIN achieves this goal through a machine learning based online adaptation of the performance interference and energy models of the system, the subsequent re-computation of optimal performance targets and a robust model predictive control based target tracking.

B. The Architecture

Figure 2 presents the architectural overview of the management components used in NINEPIN. The computer system under control is a virtualized server hosting multiple customer applications in VMs that logically abstract the resources provided by the underlying multi-core server. In case of interactive multi-tier applications, each tier of an application is deployed at a virtual machine. The NINEPIN framework forms a control loop that non-invasively mitigates the performance interference between co-located VMs by adjusting their CPU resource allocation (i.e, CPU usage limits) in an energy efficient manner, so that the overall system utility is maximized. The key components in the control loop include a two-level hierarchical controller, a power monitor, a performance monitor for each VM and a resource controller. The two-level control framework integrates utility optimization with control theoretical approach while avoiding highly complex system modeling and computationally expensive control.

1) Power and Performance Monitors: The power monitor measures the average power consumption of the underlying multi-core server for the last control interval.

The performance monitor measures the average performance of the hosted applications in the last control interval. The actual performance metrics may vary for heterogeneous applications running in virtualized environments. Our design does not use any semantic information regarding these performance metrics. It treats the performance values as raw data for modeling and control. Hence, NINEPIN is applicable to any performance metric.

2) Level-1 Control: At level-1, the utility optimizer calculates the optimal performance targets for each VM in order to maximize the overall system utility and sends the calculated targets to the level-2 controller. The optimization is based on fuzzy MIMO models that capture the performance interference relationship between co-located VMs and the energy consumption property of the underlying server for various CPU resource allocations. These models are constructed offline by applying machine learning techniques on various data collected from the system as described in Section IV-A2.

It periodically collects the values of power consumption from the power monitor, average performance of running applications from the performance monitor and the CPU usage limits on various VMs from server logs. Then, it calculates the corresponding energy usage due to various applications running in the virtualized server. The total energy usage is a product of the average power consumption and the average completion time of the longest running application.

The measured values are compared with the values of energy usage and performance predicted by the fuzzy MIMO models. If there are significant prediction errors, the fuzzy MIMO models are updated based on the new observations and the optimal performance targets are re-calculated. Such prediction errors can occur due change in workload.

3) Level-2 Control: At level-2, the model predictive controller computes the CPU usage limits to be enforced on each VM in order to track the optimal performance targets set by the utility optimizer. For this purpose, it uses a linear state-space performance interference model, which is obtained by linearizing the fuzzy MIMO model at each operating point. Linearization reduces the computational complexity of the control problem. It is designed to achieve the performance targets while maintaining system stability in spite of the inevitable uncertainties and disturbances in the system.

The CPU resource allocator is the actuator for this control system. It performs the control actions by enforcing the computed CPU usage limits on the co-located VMs in order to regulate the system towards the optimal targets. Applying CPU usage limits affects a VM's performance as well as power consumption. It is due to the idle power management of modern processors, which can achieve substantive energy savings when a processor is idle compared to it is active.

IV. Level-1 Controller Design

Level-1 control computes the optimal service levels of customer applications co-located in a single virtualized server and sends these values as the performance targets to the level-2 controller. It performs utility optimization based on the current system models. The performance interference model is for non-invasive performance isolation with heterogeneous application support and the energy usage model is for energy efficiency.

A. Performance Interference and Energy Usage Models

The performance interference and energy usage models are based on fuzzy MIMO modeling technique. In this section, we formulate a fuzzy MIMO model to represent a virtualized multi-core server system hosting multiple applications and discuss a machine learning based model construction technique. MIMO modeling is well-suited to capture the performance interference interactions between co-located VMs. Together with fuzzy logic, it accurately represents the highly complex and nonlinear relationship between various system variables. This is important for achieving modeling accuracy and self-adaptiveness of the system model at the same time. Although the initial fuzzy model is learned for a group of applications, it is adaptive to different workload mixes at run time. We discuss the need for self-adaptiveness in Section IV-C.

1) Fuzzy MIMO Model Formulation: We consider a number of applications hosted in a multi-core server as a MIMO system. The inputs to the system are CPU usage limits set for various applications. The outputs of the system are the measured performance of each application and the energy usage of the underlying server. We obtain two separate models for energy usage and performance of the system, respectively. The system is approximated by a collection of MIMO fuzzy models as follows:

$$\mathrm{y}(k+1) = \mathrm{R}(\xi(k), \mathrm{u}(k)). \tag{1}$$

Let $\mathrm{y}(k)$ be the output variable and $\mathrm{u}(k) = [u_1(k), .., u_m(k)]^T$ be the vector of current inputs at sampling interval k. The regression vector $\xi(k)$ includes current and lagged outputs:

$$\xi(k) = [\mathrm{y}(k), .., \mathrm{y}(k-n_y))]^T \tag{2}$$

where n_y specifies the number of lagged values of the output variable. Note that a regression vector may also include lagged inputs to achieve even better accuracy of energy usage and performance prediction. R is a rule based fuzzy model consisting of K fuzzy rules. Each fuzzy rule is described as follows:

R_i: If $\xi_1(k)$ is $\Omega_{i,1}$ and .. $\xi_\varrho(k)$ is $\Omega_{i,\varrho}$ and $u_1(k)$ is $\Omega_{i,\varrho+1}$ and .. $u_m(k)$ is $\Omega_{i,\varrho+m}$ then

$$\mathrm{y}_i(k+1) = \zeta_i \xi_i(k) + \eta_i \mathrm{u}(k) + \phi_i. \tag{3}$$

Here, Ω_i is the antecedent fuzzy set of the i^{th} rule which describes elements of regression vector $\xi(k)$ and the current input vector $\mathrm{u}(k)$ using fuzzy values such as 'large', 'small', etc. ζ_i and η_i are vectors containing the consequent parameters and ϕ_i is the offset vector. ϱ denotes the number of elements in the regression vector $\xi(k)$. Each fuzzy rule describes a region of the complex non-linear system model using a simple functional relation given by the rule's consequent part. The model output is calculated as the weighted average of the linear consequents in the individual rules. That is,

$$\mathrm{y}(k+1) = \frac{\sum_{i=1}^{K} \beta_i(\zeta_i \xi_i(k) + \eta_i \mathrm{u}(k) + \phi_i)}{\sum_{i=1}^{K} \beta_i} \tag{4}$$

where the degree of fulfillment for the i^{th} rule β_i is the product of the membership degrees of the antecedent variables in that rule. Membership degrees are determined by fuzzy membership functions associated with the antecedent variables. The model output is expressed in the form of

$$\mathrm{y}(k+1) = \zeta^* \xi_i(k) + \eta^* \mathrm{u}(k) + \phi^*. \tag{5}$$

The aggregated parameters ζ^*, η^* and ϕ^* are the weighted sum of vectors ζ_i, η_i and ϕ_i respectively.

$$\begin{aligned} \zeta^* &= \frac{\sum_{i=1}^{K} \beta_i \cdot \zeta_i}{\sum_{i=1}^{K} \beta_i}. \\ \eta^* &= \frac{\sum_{i=1}^{K} \beta_i \cdot \eta_i}{\sum_{i=1}^{K} \beta_i}. \\ \phi^* &= \frac{\sum_{i=1}^{K} \beta_i \cdot \phi_i}{\sum_{i=1}^{K} \beta_i}. \end{aligned}$$

2) Machine Learning Based Model Construction: We construct initial fuzzy models by applying a subtractive clustering technique on data collected from the system. Each obtained cluster represents a certain operating region of the system, where input-output data values are highly concentrated. The clustering process partitions the input-output space and determines the number of fuzzy rules and the shape of membership functions. Then, we apply an adaptive network based fuzzy inference system (ANFIS) to further tune the fuzzy model parameters. It constructs an artificial neural network to represent a fuzzy model and tunes its parameters using a combination of back-propagation algorithm with a least squares method. This adjustment allows the fuzzy system to learn from the data it is modeling. The data set includes various values of energy usage and performance measured from the system for past resource allocations.

B. Utility Optimizer

The utility optimizer is responsible for finding the optimal service level for each application so that the overall system utility is maximized. It maintains the knowledge about the service-level utility function for each application, the utility function of energy consumption, the performance interference model of co-located VMs and the energy usage model of the underlying multi-core server. The service-level utility function reflects the revenue or penalty related to service-level agreements with customers, and may also incorporate additional considerations such as the value of maintaining the data center's reputation for providing good service. It is of the form $U_i(S)$ for application i, where S is the service level achieved in terms of its average performance. The energy utility $U(E)$ represents the costs associated with energy consumption. For a given combination of CPU resource allocations to various co-located applications, the performance interference model specifies the service levels that each application can achieve and the energy usage model specifies the energy consumption of the virtualized server.

The optimization problem is formulated as follows:

$$Maximize \sum_{i=1}^{N} U_i(S) + \epsilon * U(E) \tag{6}$$

where ϵ is a tunable coefficient expressing the relative value of energy efficiency and performance objectives. The utility optimizer first computes the CPU usage limits to be enforced on co-located VMs so that the overall system utility given by Eq. (6) is maximized. Then, it determines the optimal service level targets for each application corresponding to the computed CPU usage limits by using the performance interference model. Our optimization algorithm is shown in Algorithm 1.

Various heuristic optimization algorithms such as Simulated Annealing, Genetic Algorithm, Hill Climbing and Particle Swam are well-suited to the optimization problem, due to the complex non-linear relationship between the objective function and the decision variables. Here, the decision variables are the CPU usage limits to be enforced on co-located VMs. In this work, we apply a genetic algorithm based technique that searches the space of various possible CPU usage limits and finds a near-to-optimal solution. It uses the negative of utility optimization objective in Eq. (6) as the *fitness function* since the genetic algorithm is designed to minimize the fitness function. As a result, it maximizes the system utility. We represent a solution to the optimization problem by a chromosome. It is a string of numbers, coding information about the decision variables. The genetic algorithm generates a new population of candidate solutions and evaluates their fitness values in various iterations. We observe that it is able to converge within 600 iterations or generations.

Algorithm 1 The optimization algorithm.

```
1: Start with a random initial population where each individual represents a combination of CPU usage limits on co-located VMs.
2: repeat
3:     Evaluate each individual solution's fitness according to the defined fitness function.
4:     Select pairs to mate from best-ranked individuals based on their fitness scores.
5:     Apply crossover and mutation operations on the selected pairs to generate a new population.
6: until Number-of-generations $\leq G$
7: Calculate the optimal performance targets corresponding to the final solution of CPU usage limits, based on the performance interference model.
```

C. Online Model Adaptation for Robust Performance Isolation

NINEPIN provides robust performance isolation with heterogeneous application support. It addresses the practical issue of hosting compute intensive and interactive applications in the same virtualized server in two steps. First, it performs online adaptation of the performance interference and energy usage models in response to a significant workload variation of interactive applications that are co-located with VMs running compute intensive jobs. Then, it re-computes the optimal performance targets corresponding to the updated system model. Furthermore, the model predictive controller performs control actions based on the updated performance interference model. Hence, NINEPIN is robust against variations in the workload and heterogeneity of the hosted applications.

The online model adaptation is performed only when a significant error in the prediction of energy usage and performance is detected. This avoids the overhead of frequent adaptation and computationally expensive re-optimization. NINEPIN applies a wRLS (weighted Recursive Least Squares) method to adapt the consequent parameters of the fuzzy MIMO model as new measurements are sampled from the runtime system. It applies exponentially decaying weights on the sampled data so that larger weights are assigned to more recent observations.

For online model adaptation, we express the fuzzy model output in Eq. (4) as follows:

$$\mathrm{y}(k+1) = X\theta(k) + e(k) \tag{7}$$

where e(k) is the error value between actual output of the system (i.e., measured performance) and predicted output of the model. $\theta = [\theta_1^T \theta_1^T .. \theta_p^T]$ is a vector composed of the model parameters. $X = [w_1 X(k), w_2 X(k), .., w_p X(k)]$ where w_i is the normalized degree of fulfillment of i^{th} rule and $X(k) = [\xi_i^T(k), \mathrm{u}(k)]$ is a vector containing current and previous outputs and inputs of the system. The parameter vector $\theta(k)$ is estimated so that the following cost function is minimized. That is,

$$Cost = \sum_{j=1}^{k} \lambda^{k-j} e^2(j). \tag{8}$$

Here λ is a positive number smaller than one. It is called "forgetting factor" as it gives larger weights on more recent samples in the optimization. This parameter determines in what manner the current prediction error and old errors affect the update of parameter estimation. The parameters of fuzzy model are updated by the wRLS method.

$$\theta(k) = \theta(k-1) + Q(k)X(k-1)[y(k) - X(k-1)\theta(k-1)]. \tag{9}$$

$$Q(k) = \frac{1}{\lambda}[Q(k-1) - \frac{Q(k-1)X(k-1)X^T(k-1)Q(k-1)}{\lambda + X^T(k-1)Q(k-1)X(k-1)}]. \tag{10}$$

$Q(k)$ is the updating matrix. The initial value of $\theta(0)$ is equal to the value obtained in the off-line identification.

V. Level-2 Controller Design

The level-2 controller applies the model predictive control principle to regulate the virtualized server system's dynamic behavior towards the optimal performance targets. The main advantage of using a control theoretical foundation is the ability to achieve the performance targets with better control accuracy and stability in spite of the inevitable uncertainties and disturbances that exist in the system.

A. Linearized State-Space Model

To apply the model predictive control theory on the virtualized server system, the level-2 controller first linearizes the fuzzy MIMO model and represents it as a state-space linear time variant model in the following form:

$$\begin{aligned} x_{lin}(k+1) &= A(k)x_{lin}(k) + B(k)\mathrm{u}(k). \\ \mathrm{y}(k) &= C(k)x_{lin}(k). \end{aligned} \tag{11}$$

The state vector for the state-space description is defined as

$$x_{lin}(k+1) = [\xi^T(k), 1]^T. \tag{12}$$

The matrices $A(k)$,$B(k)$ and $C(k)$ are constructed by freezing the parameters of the fuzzy model at a certain operating point $\mathrm{y}(k)$ and u(k) as follows. The current operating point is determined by the performance values of each application and the current CPU usage limits measured from the runtime system. We calculate the degree of fulfillment β_i for the current inputs (i.e CPU usage limits) chosen for the system and compute the aggregated parameters ζ^*, η^* and ϕ^*. Comparing Eq. (5) and Eq. (11), the state matrices are computed as follows:

$$A = \begin{bmatrix} \zeta^*_{1,1} & \zeta^*_{1,2} & .. & .. & .. & \zeta^*_{1,\varrho} & \phi^*_1 \\ 1 & 0 & .. & & & 0 & 0 \\ 0 & 1 & \vdots & & & 0 & 0 \\ \vdots & \vdots & \ddots & & & \vdots & \vdots \\ \zeta^*_{2,1} & \zeta^*_{2,2} & .. & .. & .. & \zeta^*_{2,\varrho} & \phi^*_2 \\ 0 & \vdots & \ddots & & & \vdots & \vdots \\ \zeta^*_{p,1} & \zeta^*_{p,2} & .. & .. & .. & \zeta^*_{p,\varrho} & \phi^*_p \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & .. & 0 & .. & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \eta^*_{1,1} & \eta^*_{1,2} & .. & \eta^*_{1,m} \\ 0 & .. & .. & 0 \\ \vdots & & & \vdots \\ \eta^*_{2,1} & \eta^*_{2,2} & .. & \eta^*_{2,m} \\ 0 & .. & .. & 0 \\ \vdots & & & \vdots \\ \eta^*_{p,1} & \eta^*_{p,2} & .. & \eta^*_{p,m} \\ 0 & .. & .. & 0 \\ 0 & .. & .. & 0 \end{bmatrix} C = \begin{bmatrix} 1 & 0 & .. & .. & .. & .. & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & .. & .. & .. & .. & 1 & 0 \end{bmatrix}$$

where ζ^*_{ij} is the j^{th} element of aggregate parameter vectors ζ^* for application i. Similarly, η^*_{ij} is the j^{th} element of aggregate parameter vectors η^* for application i.

B. The Model Predictive Controller

The control goal is to steer the system into a state of optimum target tracking, while penalizing large changes in the control variables. It minimizes the deviation of application performance from their respective targets.

1) MIMO Control Problem: The model predictive controller decides the control actions at every control period k by minimizing the following cost function:

$$V(k) = \sum_{i=1}^{H_p} ||r - y(k+i)||^2_P + \sum_{j=0}^{H_c-1} ||\Delta\mathrm{u}(k+j)||^2_Q. \tag{13}$$

Here, $y(k)$ is a vector containing the performance measure of each application. The controller uses the linearized state-space model to predict each application's performance over H_p control periods, called the *prediction horizon*. It computes a sequence of control actions $\Delta\mathrm{u}(k), \Delta\mathrm{u}(k+1), .., \Delta\mathrm{u}(k+H_c-1)$ over H_c control periods, called the *control horizon*, to keep the predicted performance close to their pre-defined targets r. The control action is the change in CPU usage limits imposed on various applications. P and Q are the weighting matrices whose relative magnitude provides a way to tradeoff tracking accuracy for better stability in the control actions.

The control problem is subject to the constraint that the sum of CPU usage limits assigned to all applications must be bounded by the total CPU capacity of the physical server. The constraint is formulated as:

$$\sum_{j=1}^{M} (\Delta u_j(k) + u_j(k)) \leq U_{max} \tag{14}$$

where M is the number of applications hosted in a resource pool and U_{max} is the total CPU capacity of the resource pool.

2) Transformation to Quadratic Programming Problem: We transform the control formulation to a standard quadratic programming problem, which allows us to design and implement the control algorithm based on an effective quadratic programming method. The MIMO control problem defined by Eq. (13) is transformed to a quadratic program:

$$\text{Minimize } \frac{1}{2}\Delta\mathrm{u}(k)^T H \Delta\mathrm{u}(k) + c^T \Delta\mathrm{u}(k) \tag{15}$$

subject to constraint $\Omega\Delta\mathrm{u}(k) \leq \omega$.

The matrices Ω and ω are chosen to formulate the constraints on CPU resource usage. Here, $\Delta\mathrm{u}(k)$ is a matrix containing the CPU usage limits on each virtual machine over the entire control horizon H_c. In the minimization formulation,

$$H = 2(R_u^T P R_u + Q). \tag{16}$$

$$c = 2[R_u^T P^T (R_x A x(k) - r)]^T. \tag{17}$$

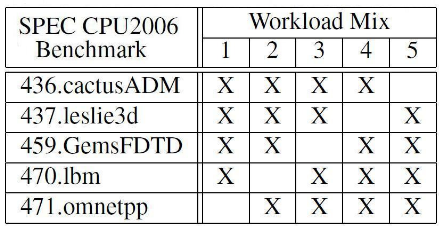
The matrices R_u and R_x are associated with the performance interference model of the hosted applications.

$$R_u = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{H_p-1} \end{bmatrix} R_x = \begin{bmatrix} CB & 0 & .. & 0 \\ CAB & CB & .. & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{H_p-1}B & CA^{H_p-1}B & .. & CA^{H_p-H_c}B \end{bmatrix}$$

VI. System Implementation

A. The Testbed

We have implemented NINEPIN on a testbed of an HP ProLiant BL460C G6 blade server module and an HP EVA storage area network with 10 Gbps Ethernet and 8 Gbps Fibre/iSCSI dual channels. The blade server is equipped with Intel Xeon E5530 2.4 GHz quad-core processor and 32 GB PC3 memory. Xeon 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 cache. Virtualization of the server is enabled by an enterprise-level virtualization product, VMware ESX 4.1. VMware's vSphere module controls the CPU usage limits in MHz allocated to the



SPEC CPU2006 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

Fig. 3. SPEC CPU2006 workload mixes.

VMs. It also provides an API to support the remote management of VMs. We create a resource pool from the virtualized server to host multiple applications. Each application is hosted inside a VMware virtual machine with one VCPU, 4 GB RAM and 15 GB hard disk space. We assign the CPU affinity of each VM to a particular CPU core. The guest operating system used is Ubuntu Linux version 10.04.

Our testbed utilizes four VMs on the same quad-core processor to host a set of four CPU bound benchmark applications from the SPEC CPU2006 suite. We choose five of the SPEC CPU2006 benchmarks that are identified as being cache sensitive in study [20], and use all possible combinations of four as experimental workload mixes shown in Figure 3.

For experiments with heterogeneous application environment, we use SPEC CPU2006 benchmark with the popular RUBiS benchmark [6], [14], [22]. RUBiS is an open-source multi-tier application benchmark. It implements the core functionality of an eBay like auction site: selling, browsing and bidding. The application contains a Java-based client that generates a session-oriented workload. RUBiS sessions have an average duration of 15 minutes and the average think time is five seconds. We use three VMs to host a three-tier RUBiS application and the fourth VM to host one SPEC CPU2006 benchmark application. We instrument RUBiS clients to generate workloads of time-varying intensity.

B. NINEPIN Components

We implement the components of the NINEPIN framework on a separate machine and issue commands to the virtualized server over the network using VMware vSphere API 4.1.

1) Power Monitor: The average power consumption of the virtualized server is measured at the resource pool level by using a new feature of VMware ESX 4.1. VMware gathers such data through its Intelligent Power Management Interface sensors. The power monitor program uses vSphere API to collect the power measurement data.
2) Performance Monitor: It uses a sensor program provided by RUBiS client for performance monitoring of the interactive application in terms of average end-to-end request response time. For compute intensive jobs, it measures the average job completion time of each VM running the SPEC CPU2006 benchmark application.
3) Performance Interference and Energy Usage Modeling: It applies subtractive clustering and ANFIS techniques on the data collected from the virtualized server system to construct performance interference and energy usage models. The fuzzy logic toolbox in MATLAB is invoked for this purpose.
4) Hierarchical Controller: It applies a genetic algorithm for system utility optimization and invokes a quadratic programming solver, *quadprog*, in MATLAB to execute the control algorithm described in Section V-B. The solution of the control algorithm in terms of VM CPU usage limits is sent to the resource allocator.
5) Resource Allocator: It uses vSphere API to impose CPU usage limits on the VMs. The vSphere module provides an interface to execute a method $ReconfigVM\_Task$ to modify a VM's CPU usage limit.

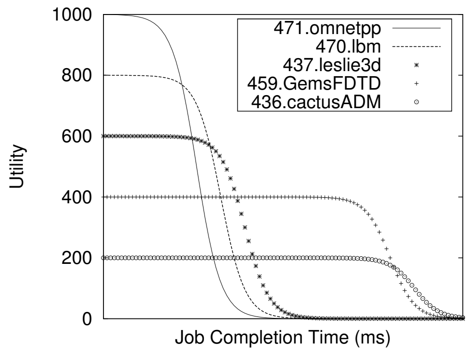


Fig. 4. Service-level utility of various SPEC CPU2006 applications.

VII. Performance Evaluation

For performance evaluation, we consider various service-level utility functions of SPEC CPU2006 applications as shown in Figure 4. We chose these utility functions as a case study without any loss of generality. We consider that the utility of energy consumption is given by a linear utility function as follows:

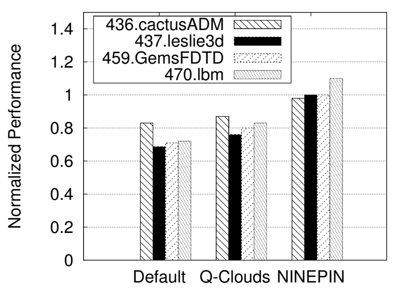
$$U(E) = \epsilon * Energy \qquad (18)$$

where *Energy* is the total energy consumed by virtualized resource pool hosting multiple SPEC CPU2006 applications and ϵ is a negative constant, which expresses the relative value of energy and performance objectives. Note that the applicability of NINEPIN in virtualized servers is independent of the chosen utility functions.

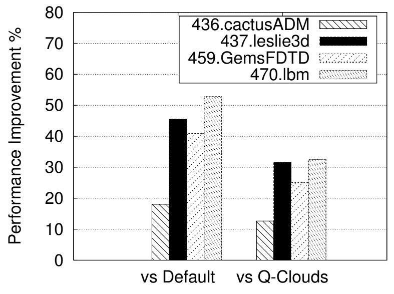
We use SPEC CPU2006 suite's *runspec* tool to run various benchmarks simultaneously on the co-located VMs. Each benchmark is run on multiple iterations to measure its average performance in terms of the SPECspeed metric. It is amount of time taken to complete a single task. The energy usage is measured in kilojoules (kJ). It is a product of the average power consumption and the average task completion time of the benchmark with the longest running tasks.

A. Performance Isolation

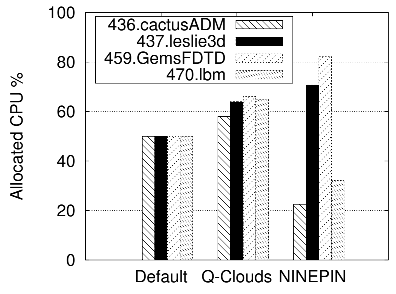
First, we study the impact of performance interference between co-located applications on their performance. We



(a) Normalized performance.



(b) Performance improvement.



(c) Resource allocations.

Fig. 5. Performance isolation by default, Q-Clouds and NINEPIN.

TABLE 1
PERFORMANCE OF WORKLOAD MIX-1'S CPU2006 BENCHMARK APPLICATIONS WITHOUT PERFORMANCE ISOLATION.

Application	Completion Time (ms)	
	Running alone	Running in co-located VMs
436.cactusADM	2433	2931
437.leslie3d	1091	1586
459.GemsFDTD	1190	1676
470.lbm	825	1145

TABLE 2
PERFORMANCE OF WORKLOAD MIX-1'S CPU2006 BENCHMARK APPLICATIONS WITH PERFORMANCE ISOLATION.

Application	Completion Time (ms)	
	Q-Clouds	NINEPIN
436.cactusADM	2796.5	2482.65
437.leslie3d	1435.52	1091
459.GemsFDTD	1487.5	1190
470.lbm	993.97	750

consider SPEC CPU2006 workload mix 1 that consists of CPU 2006 benchmark applications 436.cactusADM, 437.leslie3d, 459.GemsFDTD and 470.lbm. Table 1 compares the average completion time of the four benchmark applications in the workload mix 1 when each is run on an isolated VM as opposed to when all of them are run simultaneously in co-located VMs. All four benchmark applications exhibit performance degradation in the absence of a performance isolation mechanism. For example, for benchmark application 436.cactusADM, the performance degradation is about 20%. For benchmark application 437.leslie3d, the performance degradation is about 20%. We observe an average performance degradation of 36% in the workload mix 1 due to the fact that VMs running on adjacent CPU cores experience contention of the underlying resources.

Next, we use workload mix 1 to evaluate the performance isolation effectiveness of NINEPIN when the service-level utility and energy utility functions are not available. In this case, NINEPIN aims to mitigate the performance interference effects without optimizing the overall system utility. Table 2 show that the average completion time of each benchmark application running on co-located VMs is reduced by NINEPIN, compared to both Q-Clouds and the default case in Table 1 that does not apply any performance isolation mechanism.

TABLE 3
PERFORMANCE TARGETS FOR SPEC CPU2006 APPLICATIONS.

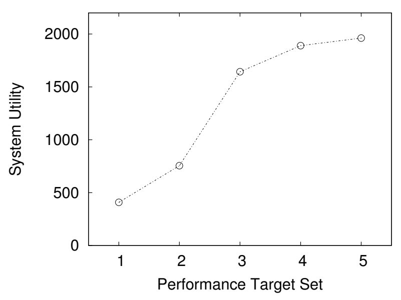
Target	CPU Equivalent Performance %			
Set	436.cactusADM	437.leslie3d	459.GemsFDTD	470.lbm
1	25	25	25	25
2	50	50	50	50
3	75	75	75	75
4	100	100	100	100
5	70	83	78	88

We assume that all four VMs require 50% of the CPU resource when there is no performance interference. We measure performance isolation in terms of the normalized performance of the VMs when they are co-located in the same virtualized server. The normalization is performed with respect to the performance shown by the VMs when they run in isolation. Figures 5(a) and 5(b) show that NINEPIN, compared to Q-Clouds and the default case that does not apply any performance isolation mechanism, is able to achieve much better performance isolation among co-located VMs. Figure 5(c) shows the CPU resources allocated to mitigate the performance interference between various hosted applications. The improvement in performance isolation by NINEPIN is due to the use of the fuzzy MIMO model, which captures the performance interference relationship more accurately. Due to the space limitation, we omit the results of other workload mixes and refer to the studies with workload mix 1 as the representative.

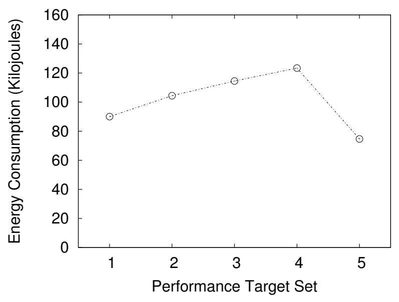
B. Optimal Performance Targeting

We evaluate the merits of utility optimization based performance targeting by NINEPIN. We define a *performance target set* as a group of performance targets for the applications co-located in a virtualized server. Each performance target is specified as the desired CPU equivalent performance that is the percentage of CPU resource required to achieve a certain performance level [21].

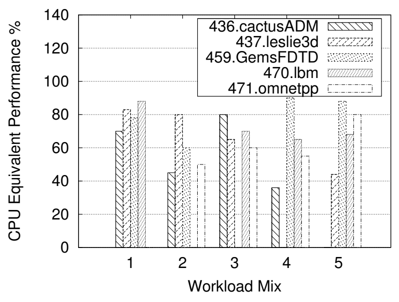
We consider SPEC CPU2006 workload mix 1 that consists of benchmark applications 436.cactusADM, 437.leslie3d, 459.GemsFDTD and 470.lbm. Table 3 gives the performance target sets. Figures 6(a) and 6(b) compare the system utility



(a) System utility.

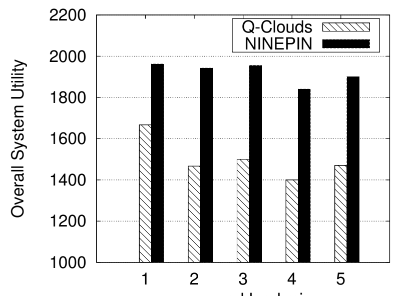


(b) Energy consumption.

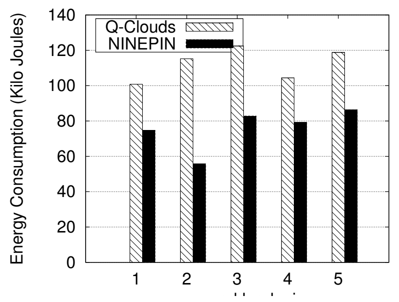


(c) Variation in optimal target.

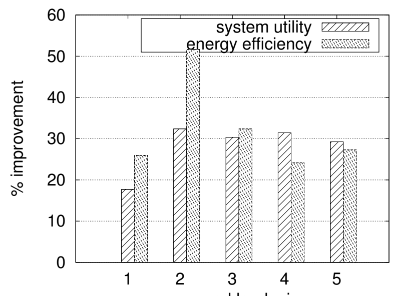
Fig. 6. Optimal performance targets achieved by NINEPIN.



(a) System utility.



(b) Energy consumption.



(c) Improvement.

Fig. 7. System utility and energy efficiency comparison between Q-Clouds and NINEPIN.

TABLE 4
UTILITY AND ENERGY EFFICIENCY.

Workload Mix	System Utility		Energy Consumption (KJ)	
	Q-Clouds	NINEPIN	Q-Clouds	NINEPIN
1	1667	1961	100.8	74.67
2	1467	1942	115.2	55.72
3	1500	1955	122.4	82.8
4	1400	1840	104.4	79.2
5	1470	1900	118.8	86.4

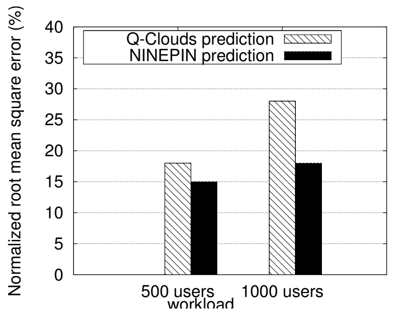
and energy consumption associated with the performance target sets. We observe that the performance target set 5, which is computed with NINEPIN, is able to maximize the system utility and minimize the energy consumption. Furthermore, Figure 6(c) shows that the optimal performance targets vary with all five different SPEC CPU2006 workload mixes. It is due to the variation in performance interference relationship and the service-level utility functions corresponding to the applications of different workload mixes. Nevertheless, NINEPIN is able assure the optimal performance targets.

C. System Utility and Energy Efficiency

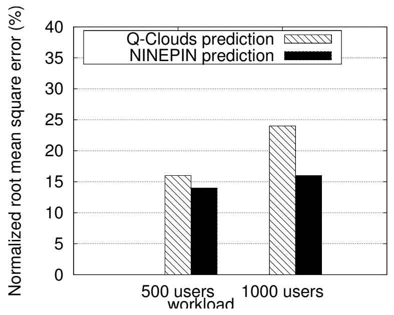
A utility based model provides a practical way to integrate performance assurance and energy efficiency goals of data center applications to maximize the profitability of cloud service provider. Table 4 compares the overall system utility and energy efficiency between NINEPIN and Q-Clouds for various workload mixes of SPEC CPU2006 suite. Note that both approaches can mitigate performance interference between co-located applications. However, NINEPIN provides significantly lower energy consumption while improving the system utility.

As shown in Figures 7(a) and 7(c), NINEPIN is able to achieve better system utility than Q-Clouds for all five workload mixes of SPEC CPU2006 suite. The system utility is a combination of the service-level utility of various co-located applications and the utility of energy consumption. NINEPIN hierarchical control framework maximizes the overall system utility by finding the optimal performance targets based on utility optimization and regulating the system to achieve the reference targets using the model predictive controller. The regulatory action takes place in the form of CPU resource allocations to co-located VMs in the virtualized server. The system utility and energy consumption varies with workload mixes. It is because different combination of applications co-located in a virtualized server manifest different performance interference relationships, energy consumption patterns and service-level utility functions. On average, the improvement in the system utility by NINEPIN is 28%.

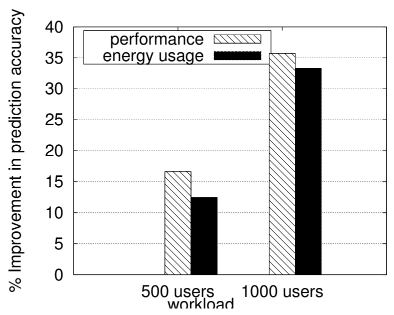
Figures 7(b) and 7(c) illustrate the improvement in energy efficiency by NINEPIN for various workload mixes of SPEC CPU2006 suite. NINEPIN reduces the energy consumption of the virtualized server by controlling each VM's CPU usage limits according to an energy usage model. It is able to tradeoff the utility of meeting performance objectives with energy efficiency. On the other hand, Q-Clouds always aims to achieve a fixed performance target without considering the cost of energy consumption. On average, the improvement in



(a) Performance.

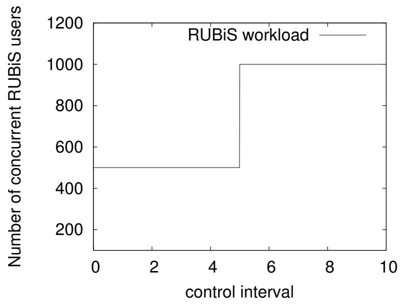


(b) Energy usage.

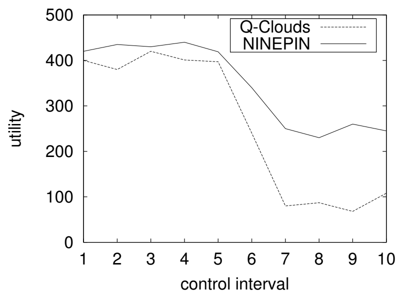


(c) Improvement.

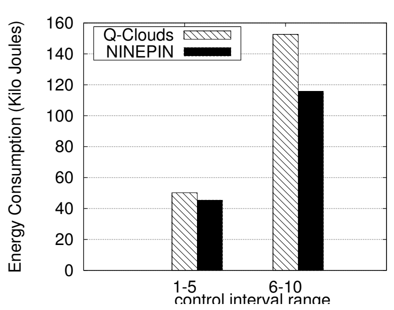
Fig. 8. Prediction accuracy of NINEPIN's performance and energy usage models.



(a) Dynamic workload.



(b) System utility.



(c) Energy Consumption.

Fig. 9. NINEPIN robustness in the face of heterogeneous applications and dynamic workload variation.

TABLE 5
IMPROVEMENT IN SYSTEM UTILITY AND ENERGY EFFICIENCY.

	Compared with	RUBiS workload	
		500 clients	1000 clients
System Utility	Default	20%	160%
	Q-Clouds	7.3%	137%
Energy Efficiency	Default	12%	27%
	Q-Clouds	9.7%	23%

energy efficiency by NINEPIN over Q-Clouds is 32% running the SPEC CPU2006 mixes.

D. NINEPIN Robustness

We evaluate the robustness of NINEPIN against application heterogeneity and dynamic workload variation. As a case study, we run one interactive three-tier application RUBiS with a dynamic workload and one SPEC CPU2006 benchmark application, 470.lbm, in the same virtualized server. RUBiS application initially faces a workload of 500 concurrent users. At the fifth control interval, the workload intensity is doubled.

The prediction accuracy of NINEPIN's system models in the face of dynamic workload variation has a significant impact on its robustness. Thus, we first measure the accuracy of the fuzzy MIMO models obtained by NINEPIN for performance and energy usage prediction. The accuracy is measured by the normalized root mean square error (NRMSE), a standard metric for deviation. We compare our results with the modeling technique used in Q-Clouds.

Figures 8(a), (b) and (c) show that NINEPIN outperforms Q-Clouds in predicting the performance of co-located applications and the energy usage of the underlying server under different workload intensities. The average improvement in the prediction accuracy of performance and energy usage are 26% and 23% respectively. The improvement is more significant when the workload changes from 500 concurrent users to 1000 concurrent users. It is due to the fuzzy MIMO model's ability to adapt more effectively to the change in workload and capture the inherent non-linearity of the system.

We measure the system utility and energy usage in the face of a dynamic workload shown in Figure 9(a). Figure 9(b) illustrates the instantaneous system behavior of the virtualized server under the influence of Q-Clouds and NINEPIN mechanisms for performance isolation. We observe that NINEPIN achieves consistently lower energy consumption and improved system utility as compared to Q-Clouds. At the fifth control interval, there is a sharp decline in the system utility for both performance isolation mechanisms. It is due to a sudden change in the performance interference relationship between heterogeneous applications, which is caused by the workload variation. Furthermore due to increase in the workload intensity, the energy consumption by the underlying server also increases. Indeed, performance interference effects are impacted by the workload intensity as well as characteristics of co-located applications. Note that the performance improvement by NINEPIN is more significant after the fifth control interval. It is due to its ability to re-compute and assure the optimal operating conditions of the system in response to the changing performance interference relationship between heterogeneous applications. Figure 9(c) summarizes the energy consumption improvement by NINPIN.

Table 5 shows the improvement in system utility and energy efficiency by NINEPIN for different RUBiS workloads, compared to Q-Clouds and the default case that does not apply any performance isolation mechanism. In the two scenarios, NINEPIN outperforms Q-Clouds in average energy efficiency and average system utility by 16% and 72%, respectively.

VIII. CONCLUSION AND FUTURE WORK

Performance isolation among heterogeneous customer applications is an important but very challenging problem in a virtualized data center. NINEPIN provides a desirable non-invasive performance isolation mechanism for a data center hosting third-party customer applications and using virtualization software from third-party vendors. As demonstrated by modeling, analysis and experimental results based on the testbed implementation, its main contributions are robust performance isolation of heterogeneous applications, energy efficiency and overall system utility optimization. It increases data center utility by aligning performance isolation goals with a data center's economic optimization objective. The main technical novelty of NINEPIN is due to the proposed and developed hierarchical control framework that integrates the strengths of machine learning based system modeling, utility based performance targeting and a model predictive control based target tracking and optimization.

Our future work will extend NINEPIN to address performance interference between I/O bound workloads.

ACKNOWLEDGEMENT

This research was supported in part by NSF CAREER award CNS-0844983.

REFERENCES

[1] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *Proc. ACM SIGMETRICS*, pages 303–314, 2005.

[2] S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In *IEEE/ACM Proc. Int'l Symposium on Microarchitecture (MICRO)*, 2006.

[3] A. Fedorova, M. Seltzer, and M. D. Smith. Improving performance isolation on chip multiprocessors via an operating system scheduler. In *Proc. Int'l Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2007.

[4] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal power allocation in server farms. In *Proc. ACM SIGMETRICS*, 2009.

[5] D. Gmach, J. Rolia, and L. Cherkasova. Resource and virtualization costs up in the cloud: Models and design choices. In *Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*, 2011.

[6] J. Gong and C.-Z. Xu. vpnp: Automated coordination of power and performance in virtualized datacenters. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, 2010.

[7] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proc. ACM/IFIP/USENIX Int'l Conference on Middleware*, 2006.

[8] C. Jiang, X. Xu, J. Wan, J. Zhang, X. You, and R. Yu. Power aware job scheduling with qos guarantees based on feedback control. In *Proc. IEEE Int'l Workshop on Quality-of-Service (IWQoS)*, 2010.

[9] G. Jung, K. Joshi, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. Performance and availability aware regeneration for cloud based multitier applications. In *Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*, 2010.

[10] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using os observations to improve performance in multicore systems. *IEEE MICRO*, 28(3):54–66, 2008.

[11] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, W. Zhihua, and C. Pu. An analysis of performance interference effects in virtual environments. In *Proc. IEEE Int'l Symposium on Performance Analysis of Systems Software (ISPASS)*, 2007.

[12] D. Kusic and J. O. Kephart. Power and performance management of virtualized computing environments via lookahead control. In *Proc. IEEE Int'l Conference on Autonomic computing (ICAC)*, 2008.

[13] P. Lama and X. Zhou. Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee. In *Proc. IEEE/ACM Int'l Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 151–160, 2010.

[14] P. Lama and X. Zhou. aMOSS: Automated multi-objective server provisioning with stress-strain curving. In *Proc. IEEE Int'l Conference on Parallel Processing (ICPP)*, pages 345–354, 2011.

[15] P. Lama and X. Zhou. PERFUME: Power and performance guarantee with fuzzy mimo control in virtualized servers. In *Proc. IEEE Int'l Workshop on Quality of Service (IWQoS)*, pages 1–9, 2011.

[16] P. Lama and X. Zhou. Efficient server provisioning with control for end-to-end delay guarantee on multi-tier clusters. *IEEE Transactions on Parallel and Distributed Systems*, 23(1), 2012.

[17] K. Le, R. Bianchiniy, M. Martonosiz, and T. D. Nguyeny. Cost- and energy-aware load distribution across data centers. In *Proc. Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[18] J. C. B. Leite, D. M. Kusic, D. Mossé, and L. Bertini. Stochastic approximation control of power and tardiness in a three-tier Web-hosting cluster. In *Proc. IEEE Int'l Conference on Autonomic computing (ICAC)*, 2010.

[19] X. Leon and L. Navarro. Limits of energy saving for the allocation of data center resources to networked applications. In *Proc. IEEE Int'l Conference on Computer Communications (INFOCOM)*, 2011.

[20] 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 *Proc. Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2008.

[21] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proc. of the 5th ACM European conference on Computer systems (EuroSys)*, 2010.

[22] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of the EuroSys Conference (EuroSys)*, pages 13–26, 2009.

[23] C. Pham, D. Chen, Z. Kalbarczyk, R. Iyer, S. Sarkar, and R. Hosn. Cloudval: A framework for validation of virtualization environment in cloud infrastructure. In *Proc. IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN)*, 2011.

[24] D. K. Tam, R. Azimi, L. B. Soares, and M. Stumm. Rapidmrc: approximating l2 miss rate curves on commodity systems for online optimizations. In *Proc. Int'l Conference on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2009.

[25] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. on Autonomous and Adaptive Systems*, 3(1):1–39, 2008.

[26] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proc. IEEE Int'l Conference on Autonomic Computing (ICAC)*, 2004.

[27] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. on Parallel and Distributed Systems*, 22(2), 2011.

[28] Y. Xie and G. H. Loh. Pipp: promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proc. Int'l Symposium on Computer architecture (ISCA)*, 2009.

[29] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *Proc. of the 4th ACM European conference on Computer systems (EuroSys)*, 2009.

[30] L. Zhao, R. Iyer, R. Illikkal, J. Moses, S. Makineni, and D. Newell. Cachescouts: Fine-grain monitoring of shared caches in cmp platforms. In *Proc. Int'l Conference on Parallel Architecture and Compilation Techniques (PACT)*, 2007.

[31] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proc. Int'l Conference on Architecture Support for Programming Language and Operating System (ASPLOS)*, 2010.