IEEE Transactions on Services Computing, Volume 15, Issue 2, March/April 2022, pp. 956-969
DOI: 10.1109/TSC.2020.2966972

Robust Dynamic CPU Resource Provisioning in Virtualized Servers

Evagoras Makridis, Kyriakos Deliparaschos, Evangelia Kalyvianaki,
Argyrios Zolotas, *Senior Member, IEEE*, and Themistoklis Charalambous, *Member, IEEE*

Abstract—We present robust dynamic resource allocation mechanisms to allocate application resources meeting Service Level Objectives (SLOs) agreed between cloud providers and customers. In fact, two filter-based robust controllers, i.e. $\mathcal{H}_\infty$ filter and Maximum Correntropy Criterion Kalman filter (MCC-KF), are proposed. The controllers are self-adaptive, with process noise variances and covariances calculated using previous measurements within a time window. In the allocation process, a bounded client mean response time (mRT) is maintained. Both controllers are deployed and evaluated on an experimental testbed hosting the RUBiS (Rice University Bidding System) auction benchmark web site. The proposed controllers offer improved performance under abrupt workload changes, shown via rigorous comparison with current state-of-the-art. On our experimental setup, the Single-Input-Single-Output (SISO) controllers can operate on the same server where the resource allocation is performed; while Multi-Input-Multi-Output (MIMO) controllers are on a separate server where all the data are collected for decision making. SISO controllers take decisions not dependent to other system states (servers), albeit MIMO controllers are characterized by increased communication overhead and potential delays. While SISO controllers offer improved performance over MIMO ones, the latter enable a more informed decision making framework for resource allocation problem of multi-tier applications.

Index Terms—Resource provisioning, virtualized servers, CPU allocation, CPU usage, RUBiS, Robust prediction, $\mathcal{H}_\infty$ filter, MCC-KF, Kalman filter.

1 Introduction

Clouds often employ multiple geographically distributed data centers to allow application deployment in various locations around the world for reduced response times. A data center contains tens of thousands of server machines located in a single warehouse to reduce operational and capital costs. Within a single data center, modern cloud applications are typically deployed over multiple servers to cope with resource requirements. The task of allocating resources to applications is referred to as *resource management*. However, resource management remains a challenge, as cloud applications often exhibit highly variable and unpredicted workload demands. To cope with the most demanding and rare workloads, over-provision of application resources has been common practice in resource management. Although simplistic, this has led to substantial under-utilization of data centers (see, e.g., [1]), since practitioners devote disjoint groups of server machines to a single application. The advent of virtualization has enabled a highly configurable environment for application deployment. A server machine can be partitioned into multiple *Virtual Machines (VMs)*, each providing an isolated server environment capable of hosting a single application or parts of it in a secure and resource assured manner. Resource allocation to VMs can be changed at runtime to dynamically match virtualized application workload demands. Virtualization enables server consolidation where a single physical server can run multiple VMs while sharing its resources and running different applications within the VMs for better utilization of the existing resources [2]. Studies show that server consolidation increases data center utilization, thus reducing energy consumption and operational costs [3].

The main challenge of server consolidation is how to dynamically adjust the allocation of VM resources so as to match the demands of virtualized applications, meet their Service Level Objectives (SLOs) and to achieve increased server utilization. Towards this end, different autonomic resource management methods have been proposed to dynamically allocate resources across virtualized applications having diverse workload and highly fluctuating workload demands. Autonomic resource management in a virtualized environment using control-based techniques has recently gained significant attention (see [4]–[7] and references therein). An advantage of such techniques over previously used heuristics is that one can derive formal guarantees [8], [9]. One common approach for controlling application performance is to control its CPU utilization within the VM.

1.1 Related Work

Works presented in [10], [11] were among the first to connect response times with the control of application CPU utilization within the VM. The use of control-based techniques has emerged as a natural approach for resource provisioning in virtualized environments, with controllers being designed to continuously update the maximum CPU allocated to each VM, based on CPU utilization measurements. For example,

- *E. Makridis and T. Charalambous are with the Dept. of Electr. Eng. and Autom., Aalto Univ. E-mail: name.surname@aalto.fi*
- *K. Deliparaschos is with the Dept. of Electr. Eng., Comp. Eng. and Inform., Cyprus Univ. of Technology. E-mail: k.deliparaschos@cut.ac.cy*
- *E. Kalyvianaki is with the Dept. of Comp. Sci. and Tech., Univ. of Cambridge. E-mail: ek264@cam.ac.uk*
- *A. Zolotas is with the Sch. of Aerosp., Transp. and Manufact., Cranfield University. E-mail: a.zolotas@cranfield.ac.uk*

Padala *et al.* [12] present a two-layer non-linear controller to regulate the virtualized component utilization of multi-tier applications. In [13] and [10], the authors use *offline* system identification to directly control the application response times through runtime resource CPU allocation. In particular, the relationship between the response times and the CPU allocations, in regions where it is measured to be linear, is modelled. However, as this relationship is application-specific and relies on offline identification performance models, it cannot be applied when multiple applications are running concurrently, nor can it be readily adjusted to new conditions. Maggio *et al.* in [14] propose a simple dynamic heartbeat rate model for automating core allocation based on deadline metrics from the application to meet the desired goal specified. Specifically, they introduce a control scheme using the dynamic heartbeat rate model to monitor and control computer software applications by expressing and measuring the error from their desired goals. In contrast to [14], a budget constrained approach is proposed by Zhu and Agrawal in [15], in which they develop a feedback control based approach for maximizing the Quality of Service (QoS) of an application while keeping within specified time and resource budget constraints.

However, for the dynamical parameter adaptation of their feedback control, the authors used an application-specific model. Another similar control technique was proposed by the authors in [16] i.e., a mechanism for monitoring and auto-scaling of resources at the VM and container level using a discrete-time feedback controller. Lakew *et al.* in [17] proposed two different models (i.e., queue length based and inverted response time) to detect and allocate the appropriate capacity required for virtualized applications. Work in [18] used a layered queueing model which describes the relationship between resource allocation and application's performance for adding or removing resources to meet user-defined SLOs. In particular, a feedback controller based on the aforementioned model was used to dynamically adjust the number of vCPUs for single VMs (thus they did not consider a MIMO case capturing inter-resource couplings between VMs). Work in [19] presented an elasticity controller which monitors resource usage of a running VM via Auto-Regressive (AR) model proposed in [20] that predicts the next CPU usage. Although the scaling approach presented for both single VM and for two VMs, independent controllers were used not considering resource allocation interplay between the two tiers.

MIMO feedback controllers have also been considered; see, for example, [21] and [22]. These controllers make global decisions by coupling the resource usage of all components of multi-tier server applications. In addition, the resource allocation problem across consolidated virtualized applications under conditions of contention have been considered in [12], [23]: when some applications demand more resources than physically available, then the controllers share the resources among them, while respecting the user-given priorities. Kalyvianaki *et al.* [22], [24] were the first to formulate the CPU allocation problem as a state prediction one and propose adaptive Kalman-based controllers to predict the CPU utilization and maintain the CPU allocation to a user-defined threshold. Even though the standard Kalman filter (KF) provides an optimal estimate when the noise is Gaussian, it may perform poorly if the noise characteristics are non-Gaussian. Another work done in [25], proposed a Key Performance Indicator (KPI)-agnostic methodology to design vertical elasticity controllers (i.e., MIMO Model Predictive Controller) to adjust cloud resources (amount of CPU cores and memory) for applications in order to meet their predefined performance requirements.

1.2 Motivation and Contributions

Cloud service providers, however, encounter abrupt varying loads that deteriorate cloud elasticity on handling peak demands and potential unpredictable system faults and failures [26]. To anticipate abrupt workload changes in order to maintain a certain allocation headroom - above the utilization - and satisfy a given SLO, self-adaptive (e.g., [27]) and robust control techniques for CPU resource provisioning are desired. Here, we use robust filters to predict the random CPU utilizations of a two-tier virtualized server application and, therefore, provide the CPU allocations needed dynamically to satisfy a certain upper bound on the mRT. In particular, two controllers for the state estimation of the CPU resources are proposed: (a) an $\mathcal{H}_\infty$ filter to minimize the maximum error caused by the uncertainties in the model and (b) the Maximum Correntropy Criterion Kalman Filter which measures the similarity of two random variables using information of high-order signal statistics. The paper's contributions are summarized below:

- An adaptive $\mathcal{H}_\infty$ filter and MCC-KF (SISO and MIMO) to improve robustness in the state estimation problem. The filters track CPU resource utilization and adapt the state estimation based on previous observations and noises. Both filters are designed and evaluated via experimental setup using real-data CPU resource demands.
- A generic, dynamic CPU allocation is presented to illustrate how our proposed approaches address resource provisioning in virtualized servers. It is demonstrated that this type of controllers show improved performance under saturation periods and sudden workload changes.

1.3 Organization

The rest of the paper is organized as follows. Section 2 introduces client mean request response times and system performance assessment metric. Section 3 discusses the model for characterizing CPU utilization dynamics. The proposed robust controllers are discussed in Section 4, and the experimental setup given in Section 5. Control performance, compared with other state-of-the-art solutions and statistical analysis are presented in Section 6. Section 7 highlights remarks of the proposed SISO and MIMO controllers, while conclusions and future research direction are presented in Section 8.

2 Server Application Performance

One of the most widely used metrics for measuring server performance is the *client mean request response times* (mRT). It is difficult to predict the values of the mRT of server applications across operating regions, and different applications and workloads. However, it is known to have certain characteristics [28]. In particular, its values can be divided into three regions:

(a) when the application has abundant resources and, therefore, all requests are served as they arrive and the response times are kept low;
(b) when the utilization approaches 100% (e.g. around 70-80%) the mRT increases above the low values from the previous region, due to the fact that there are instances in which the requests increase abruptly;
(c) when resources are scarce and very close to 100%, since requests compete for limited resources, they wait in the input queues for long and, as a result, their response times increase dramatically to relatively high values.

In this work, the response time of every type of request was captured calculating the time difference between the request and its response, as Fig. 1 shows. All requests were issued to our RUBiS cluster and specifically to the web server, through the Client Emulator that was deployed on a separate physical machine. When all requests were completed, a mean value of the response times of the requests within a time interval of 1s was calculated in order to have an estimate of the mRT over time. Note that in the results for the experiments presented in Section 6, the mRT is smoothed over the sampling/control interval.

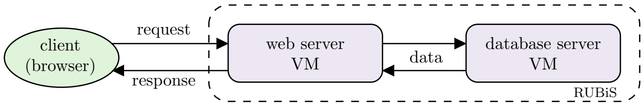


Fig. 1: Request-to-response path.

To maintain a good server performance, the operators aim for CPU utilization below 100% of the machine capacity by a certain value (usually called *headroom*). Headroom values are chosen such that they form the boundary between the second and third mRT regions. At such values the server is well provisioned and response times are kept low. If the utilization exceeds the boundary due to increased workload demands, operators should increase server resources.

Firstly, we measure the server's performance when 100% of resources is provisioned, without any controller adjusting the allocation of resources, in order to extract what is the required headroom. In this work, we consider a Browsing Mix workload type, in order to specify the server's performance while the number of clients varies. The top plot of Fig. 2 shows the mean response times (mRT) with number of clients increasing in steps of 100 until mRT crosses the 0.5s level. Clearly, the mRT increases rapidly when the number of clients exceeds 1350 and the SLO is violated.

Initially, with an increasing number of clients, the mRT stays low, albeit when the number of clients exceeds 1200 the mRT increases above the low values. Note that the QoS threshold of 0.5s is exceeded when the number of clients, simultaneously issue requests to the server is about 1350.

The bottom plot of Fig. 2 shows the average CPU usage per component while the number of clients increases. As shown in this figure, the database server demand is lower than the web server's one with the same number of clients. The error bars in the bottom plot of Fig. 2 show one standard deviation above and below the mean CPU usage. When the number of clients exceeds 1350, the web server's CPU usage becomes the bottleneck and even though the database server does not use 100% of its resources, it remains (almost) constant. Hence, it is important to establish the required resources for all the involved components comprising the requests.

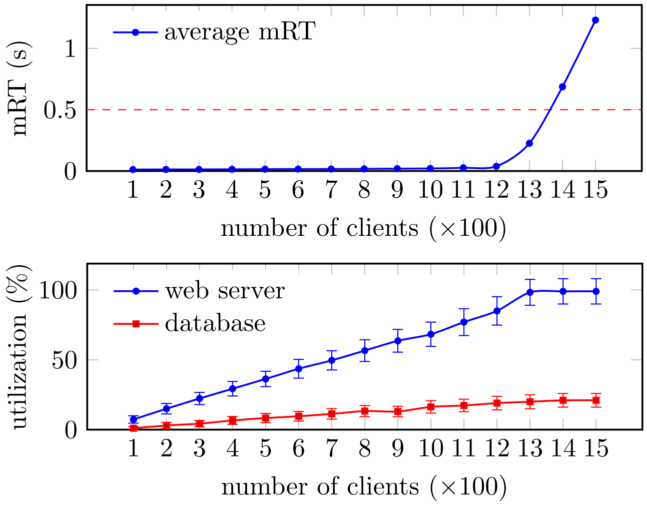


Fig. 2: *Top*: mean Response Times (mRT), *bottom*: average CPU usage per component; for different workloads.

3 System Model

3.1 Notation

Note that $\mathbb{R}$ and $\mathbb{R}_+$ represent the real and the nonnegative real numbers, respectively. Vectors, matrices and sets, are denoted by lowercase, uppercase and calligraphic uppercase letters, respectively. A^T and A^{-1} denote the transpose and inverse of matrix A, respectively. The identity matrix is represented by I. Also, $\hat{x}_{k|k-1}$ and $\hat{x}_{k|k}$ denote the *a priori* and *a posteriori* estimates of random value/vector x_k for time instant k. P_k denotes the matrix P at time instant k. $\mathbb{E}\{\cdot\}$ represents the expectation of its argument. Given any vector norm $\|\cdot\|$, a weighted vector norm can be written as $\|x\|_Q \triangleq \|Qx\|$, where Q is an arbitrary nonsingular matrix.

3.2 SISO System

The time-varying CPU utilization per component is modeled as a random walk given by the following linear stochastic difference equation as introduced in [22], [24], [29], [30]:

$$x_{k+1} = x_k + w_k, \qquad (1)$$

where $x_k \in [0,1]$ is the CPU utilization (i.e., the percentage of the total CPU capacity actually used by the application component during time interval k). The independent random process w_k is the *process noise* which models the utilization between successive intervals caused by workload changes (e.g., requests being added to or removed from the server) it is often assumed to be normally distributed [22], [24], but it can also be a distribution of finite support [29]. A similar simple model was introduced in [14], where the authors try to keep the model as simple and generic as possible, and improve the control performance by means of feedback control. In contrast, the authors in [31] adopted the autoregressive-moving-average (ARMA) model using system identification to find the relationship between mRT and the total CPU allocation in a multi-tier application. However, in this work we keep the simple model given in (1), without making any application-specific assumptions, as it is the case in [31].

The total CPU utilization of a VM which is actually *observed* by the Xen Hypervisor, $y_k \in [0,1]$, is given by

$$y_k = x_k + v_k, \tag{2}$$

where the independent random variable v_k is the utilization *measurement noise* which models the utilization difference between the measured and the actual utilization; v_k, as it is the case with w_k, is often assumed to be normally distributed [22], [24], but it can also be a distribution of finite support [29]. Note that y_k models the observed utilization in addition to any usage noise coming from other sources, such as the operating system, to support the application.

3.3 MIMO System

For the MIMO system, the dynamics of all the components (VMs) can be written compactly as

$$x_{k+1} = Ax_k + w_k, \tag{3a}$$
$$y_k = Cx_k + v_k, \tag{3b}$$

where $x_k \in [0,1]^{n_x}$ is the system's state vector representing the actual total CPU capacity percentages used by the application components during time interval k. The *process* and *measurement* noise vectors, $w_k \in \mathbb{R}^{n_x}$ and $v_k \in \mathbb{R}^{n_y}$, are stochastic disturbances with zero mean and finite second-order matrices W_k and V_k, respectively. The observed state x_k of the system by the Xen Hypervisor is $y_k \in \mathbb{R}^{n_y}$. Matrix A shows the interdependencies between different VMs and matrix C captures what is actually the Xen Hypervisor observing. In the case where the CPU utilizations at the VMs are independent, matrices A and C are identity matrices.

3.4 CPU Allocation

By $a_k \in \mathbb{R}_+$ we denote the CPU capacity of a physical machine allocated to the VM (i.e., the maximum amount of resources a VM can use). The purpose of a designed controller is to control the allocation of the VM running a server application while observing its utilization in the VM, maintaining good server performance in the presence of workload changes. This is achieved by adjusting the allocation to values above the utilization. In other words, in all filters that we design in this paper, the resource utilization is predicted and the resource allocation is chosen such that a pre-specified headroom is maintained in order to guarantee the SLO, which in this case is to have the mRT lower than 0.5s. For each time interval k, the *desired* relationship between the two quantities is given by:

$$a_k = \max\left\{a_{\min}, \min\{(1+h)x_k, a_{\max}\}\right\}, \tag{4}$$

where $h \in (0,1)$ represents the headroom (i.e., how much extra resources are provided above the actual CPU utilization) $a_{\min}$ is the minimum CPU allocated at any given time (if allocation goes very small, then even small usage may lead to high mRT), and $a_{\max}$ is the maximum CPU that can be allocated. To maintain good server performance, the allocation a_k should adapt to the utilization x_k.

Let $\mathcal{Y}_k$ represent the set of all observations up to time k. Let the *a posteriori* and *a priori* state estimates be denoted by $\hat{x}_{k|k} = \mathbb{E}\{x_k|\mathcal{Y}_k\}$ and $\hat{x}_{k+1|k} = \mathbb{E}\{x_{k+1}|\mathcal{Y}_k\}$, respectively; hence, $\hat{x}_{k+1|k}$ is the predicted CPU utilization for time interval $k+1$. In order to approach the desired CPU allocation, as given in (4), the CPU allocation mechanism uses the prediction of the usage and is thus given by

$$a_{k+1} = \max\left\{a_{\min}, \min\{(1+h)\hat{x}_{k+1|k}, a_{\max}\}\right\}. \tag{5}$$

3.5 Computation of Variances/Covariances

To estimate the variance of the process noise at time step k, W_k, using real-data for each component, we use a sliding window approach in which the variance of the data belonging in a sliding window of size T steps at each time step k is computed. Initially, the variance is chosen based on some prior information. T steps after the process is initiated, and T CPU usages have been stored, the variance is estimated.

While the mean of a *random-walk-without-a-drift* is still zero, the covariance is non-stationary. For example, for the SISO case,

$$\begin{aligned}\mathrm{var}\{x_k\} &= \mathrm{var}\{w_{k-1} + w_{k-2} + ...\}\\ &= \mathrm{var}\{w_{k-1}\} + \mathrm{var}\{w_{k-2}\} + \ldots + \mathrm{var}\{w_0\}\\ &= W_{k-1} + W_{k-2} + \ldots W_0.\end{aligned}$$

Taking the difference between two observations (i.e., $z_k \triangleq y_k - y_{k-1}$) gives

$$z_k = x_k - x_{k-1} + v_k - v_{k-1} = w_{k-1} + v_k - v_{k-1}.$$

The variance of the difference between observations is thus

$$\mathrm{var}\{z_k\} = \mathrm{var}\{w_{k-1} + v_k - v_{k-1}\} = W_{k-1} + V_k - V_{k-1}.$$

While the experiment is running, the *last* T CPU usages are stored and used for updating the variance at each step k. Computing the variance based on the difference between observations, we get:

$$\begin{aligned}&\mathrm{var}\{z_{k-T+1} + \ldots + z_k\} = \mathrm{var}\{z_{k-T+1}\} + \ldots + \mathrm{var}\{z_k\}\\ &\quad = W_{k-T} + V_{k-T+1} - V_{k-T} + \ldots + W_{k-1} + V_k - V_{k-1}.\end{aligned}$$

The measurement noise variance V_k was set to a small value because we observed that the measurement is relatively accurate since the CPU usage is captured every 1s. In other words, our measurements of the CPU usage are relatively very close to the real ones. This fact let us pin the measurement noise variance to a fixed value (herein $V_k = 1$). As a result, the variance of the difference breaks down to

$$\mathrm{var}\{z_{k-T+1} + \ldots + z_k\} = W_{k-T} + \ldots + W_{k-1}. \tag{6}$$

Hence, using (6) and assuming that the variance does not change (much) over a time horizon T, the estimate of the variance at time k, denoted by $\widehat{W}_k^{\mathrm{SISO}}$, is given by:

$$\begin{aligned}\widehat{W}_k^{\mathrm{SISO}} &= \frac{1}{T}\mathrm{var}\{z_{k-T+1} + \ldots + z_k\}\\ &= \frac{1}{T}\left\{\frac{\sum_{t=k-T+1}^{k} z_t^2}{T} - \left(\frac{\sum_{t=k-T+1}^{k} z_t}{T}\right)^2\right\}. \end{aligned} \tag{7}$$

The process noise covariance of the components is calculated using a similar methodology *mutatis mutandis* as the variances. At this point, we need to capture each component's CPU usage and store it somewhere centrally (e.g., on the MIMO controller node) in order to compute the covariances using the sliding window approach, as before. The estimate of the covariance $\widehat{W}_k^{\mathrm{MIMO}}$ for the two components of our system is given by:

$$\begin{aligned}\widehat{W}_k^{\mathrm{MIMO}} &\overset{(a)}{=} \mathrm{cov}\{z_{1,t}, \ldots, z_{1,k} \,, z_{2,t}, \ldots, z_{2,k}\} \\ &= \left(\frac{\sum_{t=k-T+1}^{k} (z_{1,t} - \mu_{z_1})(z_{2,t} - \mu_{z_2})}{T} \right), \end{aligned} \tag{8}$$

where μ_{z_1} and μ_{z_2} denote the mean CPU usages for the web server and database server components, respectively, for a window of size T, and are given by

$$\mu_{z_1} = \left(\frac{\sum_{t=k-T+1}^{k} z_{1,t}}{T} \right) \text{ and } \mu_{z_2} = \left(\frac{\sum_{t=k-T+1}^{k} z_{2,t}}{T} \right),$$

while $z_{1,t}$ and $z_{2,t}$ denote the differences between observed CPU utilizations at time instant t and $t-1$ of the first and the second component of the application, respectively.

Remark 1. Note that the size of the sliding window, T, is chosen to be large enough so that it captures the variance of the random variable, but also it is small enough so that it can also track the change in variance due to changes in the dynamics of the requests. Numerical investigation helps in choosing the sliding window T; see Section 6.

Note also that each VM can be controlled either *locally* or via a *remote* physical machine. Using the locally controlled VM as a SISO system, the estimate of the VM's variance can be obtained, but the noise covariances with respect to other applications cannot be obtained. Using a remotely controlled VM to host the MIMO controller, the noise covariances of the whole system can be estimated via (8).

4 Controller Design

This work emphasizes robust dynamic resource provisioning that accounts for model uncertainties and non-Gaussian noise. Two robust controllers are proposed in order to predict and hence allocate the CPU resources in a realistic scenario for each VM that constitutes the RUBiS application. More specifically:

- **$\mathcal{H}_\infty$ filter:** This controller minimizes the worst-case estimation error of the CPU allocation and provides robust state estimation. It can be modeled either as a SISO filter to control a single VM or as MIMO filter to control all VMs of a multi-tier application.
- **MCC-KF:** This controller is an enhanced KF version that utilizes the Maximum Correntropy Criterion for the state estimation of the CPU resources. Note the MCC-KF measures the similarity of two random variables using information of high-order signal statistics, essentially handling cases of non-Gaussian noises (which are not directly handled by the standard KF).

4.1 $\mathcal{H}_\infty$ Filter

$\mathcal{H}_\infty$ filters, called *minimax* filters, minimize the worst-case estimation error hence facilitates better robustness for the state estimation problem. In this work, we adopt a game theoretic approach to $\mathcal{H}_\infty$ filters proposed in [32] and thoroughly described in [33, Chapter 11]. The cost function for our problem formulation is given by:

$$J = \frac{\sum_{k=0}^{N-1} \|x_k - \hat{x}_{k|k}\|_2^2}{\|x_0 - \hat{x}_{0|0}\|^2_{P_{0|0}^{-1}} + \sum_{k=0}^{N-1} \left(\|w_k\|^2_{W_k^{-1}} + \|v_k\|^2_{V_k^{-1}} \right)} \tag{9}$$

where $P_{0|0} \in \mathbb{R}^{N \times N}$, $W_k \in \mathbb{R}^{N \times N}$ and $V_k \in \mathbb{R}^{N \times N}$ are symmetric, positive definite matrices defined by the problem specifications (i.e., $P_{0|0}$ is the initial error covariance matrix, W_k and V_k are the process and measurement covariance matrices for time interval k, respectively), $\hat{x}_{k|k}$ is the estimate of the CPU allocation. The direct minimization of J in (9) is not tractable and, therefore, a performance bound is chosen (i.e., $J < 1/\theta$, $\theta > 0$) and attempt to find an estimation strategy (controller, in this case) that satisfies the bound. In our problem, the target is to keep the mRT below a certain threshold (e.g., less than 0.5s). Therefore, θ is tuned such that the desired mRT is less than a certain user-specified threshold (i.e., so that the designed controller satisfies the desired target). The choice of θ will be investigated in Section 6. Considering (9), the steady-state $\mathcal{H}_\infty$ filter bounds the following cost function:

$$J = \lim_{N \to \infty} \frac{\sum_{k=0}^{N-1} \|x_k - \hat{x}_{k|k}\|_2^2}{\sum_{k=0}^{N-1} \left(\|w_k\|^2_{W_k^{-1}} + \|v_k\|^2_{V_k^{-1}} \right)}. \tag{10}$$

Let $G_{\hat{x}e}$ be the system that has $e = [w \quad v]^T$ as its input and $\hat{x}$ as its output. Since the $\mathcal{H}_\infty$ filter makes cost (10) less than $1/\theta$ for all w_k and v_k, then according to [33, Equation (11.109)]:

$$\|G_{\hat{x}e}\|_\infty^2 = \sup_{\zeta} \frac{\|x - \hat{x}\|_2^2}{\|w\|^2_{W^{-1}} + \|v\|^2_{V^{-1}}} \leq \frac{1}{\theta}, \tag{11}$$

where ζ is the phase of $\|w\|^2_{W^{-1}} + \|v\|^2_{V^{-1}}$ comprised by the sampling time of the system and the frequency of the signals. Since we want the mRT to be less than a certain value (usually around 1 second), we have to keep the CPU usage to less than a threshold set by our mRT model. Therefore, using (11) we want:

$$\sup_{\zeta} \frac{\|D\|_2^2}{\|w\|^2_{W^{-1}} + \|v\|^2_{V^{-1}}} \leq \frac{1}{\theta}, \tag{12}$$

which is equivalent to:

$$\theta \leq \inf_{\zeta} \frac{\|w\|^2_{W^{-1}} + \|v\|^2_{V^{-1}}}{\|D\|_2^2}. \tag{13}$$

where D is a diagonal matrix with the allowable error for each component along the diagonal.

Let the *a posteriori* (updated) and *a priori* (predicted) error covariances be given by

$$\begin{aligned} P_{k|k} &= \mathbb{E}\left\{ (x_k - \hat{x}_{k|k})(x_k - \hat{x}_{k|k})^T | \mathcal{Y}_k \right\}, \\ P_{k+1|k} &= \mathbb{E}\left\{ (x_k - \hat{x}_{k+1|k})(x_k - \hat{x}_{k+1|k})^T | \mathcal{Y}_k \right\}. \end{aligned}$$

The necessary condition ensuring a positive definite $P_{k|k}$ and system stability is retained for the $\mathcal{H}_\infty$ filter is:

$$I - \theta P_{k|k-1} + C^T V_k^{-1} C P_{k|k-1} \succ 0. \tag{14}$$

To design the controller we consider inequalities (13) and (14).

The equations for the $\mathcal{H}_\infty$ filter are summarized below [33]. For the *prediction phase*:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1}, \tag{15a}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + W_k. \tag{15b}$$

For the cost function (9), the *update phase* of the $\mathcal{H}_\infty$ filter is given by:

$$K_k = P_{k|k-1}[I - \theta P_{k|k-1} + C^T V_k^{-1} C P_{k|k-1}]^{-1} C^T V_k^{-1} \tag{15c}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}) \tag{15d}$$

$$P_{k|k} = P_{k|k-1}[I - \theta P_{k|k-1} + C^T V_k^{-1} C P_{k|k-1}]^{-1} \tag{15e}$$

where K_k is the gain matrix.

The KF gain is less than the $\mathcal{H}_\infty$ filter gain for $\theta > 0$, meaning that the $\mathcal{H}_\infty$ filter relies more on the measurement and less on the system model. As $\theta \to 0$, the $\mathcal{H}_\infty$ filter gain and KF gain coincide [33]. For a comparison between Kalman and $\mathcal{H}_\infty$ filters see [34].

4.2 Maximum Correntropy Criterion Kalman Filter

Here, a new Kalman filter approach is deployed using the Maximum Correntropy Criterion (MCC) for state estimation, i.e. MCC Kalman filter (MCC-KF) [35], [36]. The correntropy criterion measures the similarity of two random variables using information from high-order signal statistics [37]–[40]. Since the KF uses only second-order signal information is not optimal if the process and measurement noises are non-Gaussian noise disturbances, e.g. shot noise or mixture of Gaussian noise.

The equations for the MCC-KF are summarized below [36]. For the *prediction phase*:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1}, \tag{16a}$$

$$P_{k|k-1} = AP_{k-1|k-1}A^T + W_k, \tag{16b}$$

and for the *update phase*:

$$L_k = \frac{G_\sigma\left(\| y_k - C\hat{x}_{k|k-1}\|_{V_k^{-1}}\right)}{G_\sigma\left(\| \hat{x}_{k|k-1} - A\hat{x}_{k-1|k-1}\|_{P^{-1}_{k|k-1}}\right)}, \tag{16c}$$

$$K_k = (P^{-1}_{k|k-1} + L_k C^T V_k^{-1} C)^{-1} L_k C^T V_k^{-1}, \tag{16d}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(y_k - C\hat{x}_{k|k-1}), \tag{16e}$$

$$P_{k|k} = (I - K_k C)P_{k|k-1}(I - K_k C)^T + K_k V_k K_k^T, \tag{16f}$$

where G_σ is the Gaussian kernel, i.e.,

$$G_\sigma(\| x_i - y_i \|) = \exp\left(-\frac{\| x_i - y_i \|^2}{2\sigma^2}\right)$$

with kernel size σ[1]. Note that L_k is called the *minimized correntropy estimation cost function* and K_k is the Kalman gain (as in the $\mathcal{H}_\infty$ filter).

4.3 Resource Provisioning Algorithm

Irrespective of which filter is being used, a generic algorithm for dynamically provisioning of CPU resources of any cloud application is given in Algorithm 1.

- **Input:** Firstly, the minimum (a_{min}) and maximum (a_{max}) allocations for the VM components, the sliding window width T for the computation of the variances and covariances as well as the θ and σ parameters for the $\mathcal{H}_\infty$ and MCC-K filters, respectively, are inputs to the system.
- **Initialization:** Initial values for the process and measurement noise matrices and for the initial error covariance matrix must be declared in advance.
- **At each time step** k, the observed utilization, using the Xen Hypervisor, is set as the control input signal.
  - **Step 1 (Variance/Covariance Computation:)** Calculates the variances and/or the covariances of T passed utilizations using the approach in Section 3.5 in order to estimate the process covariance error $\widehat{W}_k$ at each time instance k.
  - **Step 2 (Filtering):** Using the statistics from the previous step, the filter updates the state while it computes the L_k, K_k, $\hat{x}_{k|k}$ and $P_{k|k}$ for each time instance k. Right after, the filter predicts the next state of the system with computing the $\hat{x}_{k+1|k}$, $P_{k+1|k}$. With this process, the a_{k+1} is computed and it can be exported as the new allocation for the next step $k+1$.
  - **Step 3 (Output):** The new predicted allocation a_{k+1} is adapted in the appropriate VM using Xen scheduler.

Algorithm 1 Dynamic Resource Provisioning.

```
Input: $a_{\min}$, $a_{\max}$, $h$, $T$, $\theta$ (for the $\mathcal{H}_\infty$ filter), $\sigma$ (for the MCC-KF),
Initialization: $W_0$, $V_0$ ($V_k = 1\ \forall k$), $P_{0|0}$
for each time step $k$ do
    Data: $y_k$
    variances/covariances
        Compute $\widehat{W}_k$ for SISO and MIMO controllers according to (7) and (8), respectively
    filter
        Update phase:
            Compute $L_k$ (for the MCC-KF), $K_k$, $\hat{x}_{k|k}$, $P_{k|k}$
        Prediction phase:
            Compute $\hat{x}_{k+1|k}$, $P_{k+1|k}$
    CPU allocation:
        Compute $a_{k+1}$ using (5)
end for
Output: CPU allocation $a_{k+1}$.
```

5 Experimental Setup

The main target is to continuously provision each virtualized application with enough CPU resources to adequately serve its incoming requests from a variable workload. The purpose of the Dom0 component is to monitor the CPU usage by the percentage of CPU cycles of each VM running on the Hypervisor. The controller utilizes the CPU measurements in order to predict the CPU usage for the next time interval and hence determine the CPU allocation, which is then fed back to to the Hypervisor to set the new allocation.

[1] The kernel bandwidth σ serves as a parameter weighting the second and higher-order moments; for a very large σ (compared to the dynamic range of the data), the correntropy will be dominated by the second-order moment [35].

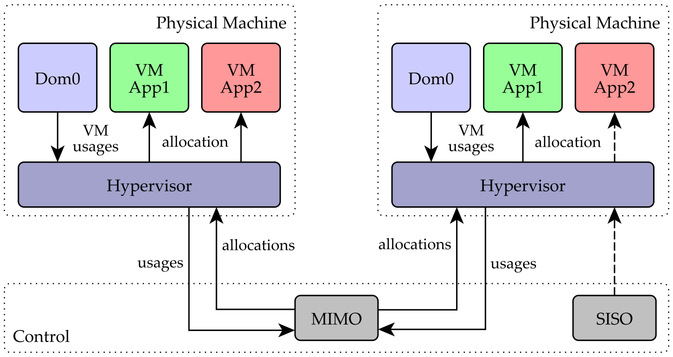


Fig. 3: Resource allocation manager architecture.

Our experimental setup is divided into two different infrastructure installations. One for the SISO controller model and one for the MIMO controller model. To evaluate the performance of each control system, we set up a small-scale data center in order to host the RUBiS auction site as the cloud application. The data center consists of two physical blade servers with Intel Xeon 5140 and 1.0GB of RAM, running Debian 8 Jessie Linux distribution with 3.16.0-4-amd64 kernel and Xen 4.4.1 Virtualization technology. Note that Xen Hypervisor was also has been widely used for experimental evaluations in the literature; for example, [22], [15] and [41]. These physical machines are used for hosting the VMs of the two-tier RUBiS benchmark application. Each physical machine, namely PM1 and PM2, hosts a VM running on Debian Jessie 8 Linux with Apache 2.4.10 web server and MySQL 5.5.55 database respectively. Note that, this infrastructure does not reflect the performance of modern servers, but it adequately serves our purpose of studying the performance of the controllers over RUBiS workload. For each physical machine we created a couple of configurations on the Xen Credit Scheduler overriding the default *time-slice* and *rate-limit* values. The default values for the Credit Scheduler are 30ms for the time-slice and 1ms for the rate-limit. We set rate-limit unchanged at its default value and time-slice at 10ms, since we determined experimentally that reducing the time-slice value increased the performance of the RUBiS application.

5.1 Benchmark Application - RUBiS

Rice University Bidding System (RUBiS), an auction site benchmark, implements the core functionality of an auction site (i.e., selling, browsing and bidding). It is modeled after ebay.com and involves a client-browser emulator, a web server, an application server and a database. It was originally used to evaluate application design patterns and application servers performance scalability. In our work, RUBiS is hosted on a two-tier application model (web server, database), while the Client Emulator generates the workload for the RUBiS application. Several RUBiS implementations exist using Java Servlets, PHP, and Enterprise Java Bean (EJB) technologies. It has provisions for selling, browsing and bidding items, allowing for different sessions for different types of users in the form of visitor, buyer and seller to be implemented. RUBiS auction site defines 26 types of actions that can be performed through the client's Web browser. In our work, the clients were modeled using the Client Emulator which is mentioned below. The MySQL database contains 7 tables used to store bids, buy now, categories, comments, items, regions and users. With cloud computing increasingly attracting the attention of researchers, RUBiS became the classic real-data benchmark for resource management problems [12], [21]–[23], [41].

5.2 Client Emulator

The Client Emulator is hosted on a third physical machine (PM3) which is dedicated for generating the RUBiS auction site workload. A Java code is responsible for generating the workload and creating user sessions to send HTTP requests to the RUBiS auction site for the purpose of emulating the client's behavior. The original version of Client Emulator provides visual statistics for throughput, response times and other information for the sessions. However, in our experiments we modified the original Client Emulator's source code in order to capture the response time of each completed request and as a next step to calculate the mRT each second or time interval. For more information about the workload generation see [42].

5.3 Resource Allocation Control

All controllers presented in this work were added on the base project code called ViResA[2], first developed for the synthetic data generation and the performance evaluation of the controllers in [30] and later for the real-data performance evaluation of the $\mathcal{H}_\infty$ and MCC-KF SISO controllers in [41]. The performance of the dynamic CPU allocation is evaluated using the mRT, which is measured at the Client-side of our prototype RUBiS server application, as the performance metric. The goal of the controllers is to adapt the CPU resource allocations of a single VM (SISO) or group of VMs (MIMO) in exchange for saving resources for other applications that are hosted on the same physical machine. An overview of the SISO system's architecture is shown in the left plot of Fig. 4. On this setup, the web server component and the database server component communicate with each other via HTTP requests, as shown in Fig. 1. Each VM of this setup is controlled in parallel via the SISO model while keeping them isolated from each other. However, changes in the demand affect both components in a relative way because each request follows a path from client to web server to database server and back, as shown in Fig. 1. Hence, the SISO system is not able to calculate the correlation between the two components. The CPU usage measurements are recorded every 1s using the Xen Hypervisor through Domain-0, which also hosts the controllers for each component. In contrast, in the MIMO case shown in the right plot of Fig. 4, a centralized controller running on a remote machine (i.e., an external physical machine (PM4) over the network), estimates the states for all the components and adjusts the allocations, while it accounts the inter-resource couplings which are calculated using the covariance computation method mentioned in Section 3.5.

[2] ViResA (Virtualized [server] Resource Allocation) is a base project code hosted in Bitbucket (https://bitbucket.org) as a private Git repository. For download requests, please contact authors.

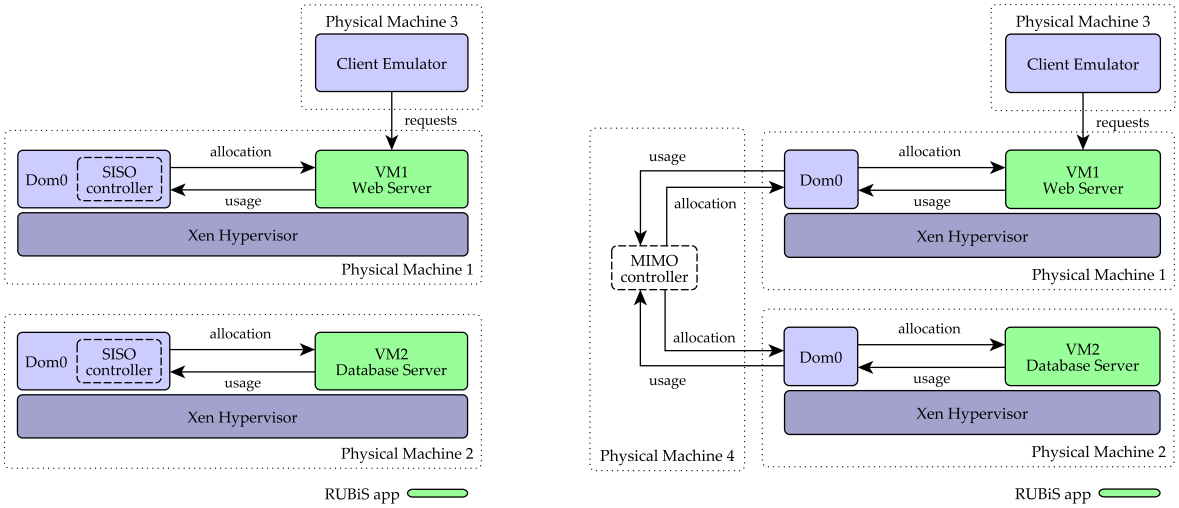


Fig. 4: System architecture (Left: SISO case, Right: MIMO case).

For both cases, the CPU usage measurements are recorded every 1s using the Xen Hypervisor through Domain-0, and the predicted allocations are updated every 5s.

6 Performance Evaluation

6.1 Evaluation Metrics

The following evaluation metrics are included for a more rigorous assessment compared to the basic performance metric (i.e., mRT).

- **Completed Requests (CR):** the total number of completed requests for all users during the full time duration of each experiment.
- **Average CPU usage:** the average CPU usage of the web server component (VM1) and database server component (VM2) for the full time duration of each experiment.
- **SLO Obedience (SLOO):** the ratio of the requests with mRT below the QoS threshold (e.g., $\mathrm{mRT} \leq 0.5$s) over the total number of completed requests (CR).
- **Average mRT (AmRT):** the average mRT for the full time duration of each experiment.

6.2 Sliding Window

As discussed in Section 3.5, the size of sliding window, T, is chosen carefully such as to be adequately large for capturing the variance of the random variable, but at the same time small enough for tracking the change in variance due to changes in the dynamics of the requests. For choosing the sliding window size T, we run a number of experiments for different values of T and observe the behavior of the variance, while recording the CPU usage of the web server component under a workload applied on RUBiS application. The workload initiates with 500 clients sending requests. At sampling point 20, another 500 clients are inserted to the workload for almost another 30 samples (one sample corresponds to 5s). Then, at sampling point 100 the number of clients sending requests to RUBiS rise up from 500 to 1500 clients for 10 samples. Finally, at sampling point 150, 1200 clients send requests to RUBiS for 30 more samples. The workload of this experiment provides a thorough CPU usage variability thus making the dynamics of the system detectable for different values of window sizes.

From the left plot of Fig. 5 it is easy to see that for $T = 1$ the change in variance is too noisy and sensitive to mild workload changes which can be observed by the abruptness and the sharpness of the standard deviation. For $T = 10$ (right plot of Fig. 5), we can see that the variance is not following the variability of the workload and if it does to a certain extent, this is delayed considerably. The best response of the variance is given for $T = 5$ (center plot of Fig. 5), in which case it is less sensitive to mild changes and it captures large and abrupt variabilities. To reduce the communication and computational overhead, for the experiments in this work we choose the window size to be 5 sampling points ($T = 5$) as it is large enough to capture the variance of the random variable and small enough to track the change in variance due to the changes in the dynamics of the requests.

6.3 Headroom

Let parameter c denote the desired CPU utilization to CPU allocation ratio (i.e., $c = 1/(1 + h)$ where h is the headroom as used in and defined right after (4)). The mRT with respect to parameter c for all the filters are shown in Fig. 6.In this evaluation, we set a stable workload of 1000 clients sending requests simultaneously to the RUBiS auction site. Each measurement is derived from experiments where c has values of 0.7, 0.8, 0.9 and 0.95 which are enough to present the behavior of mRT as parameter c grows. With $c \to 1$ more resources are available for other applications to run, but the mRT of requests is increasing which results to decreasing performance of the RUBiS benchmark (This is due to the headroom approaching 0 hence fewer resources remain for RUBiS to use). Referring to Fig. 6, both SISO and MIMO controllers can allocate resources without significant increase of mRT when $c < 0.8$. Note the mRT grows exponentially once $c > 0.8$ (however, mRT values are relatively low due to conducting the current experiment with a static number of clients). Note that: (i) a higher static number of clients sending requests to RUBiS does not always lead to a higher mRT value; (ii) workloads with relatively smooth

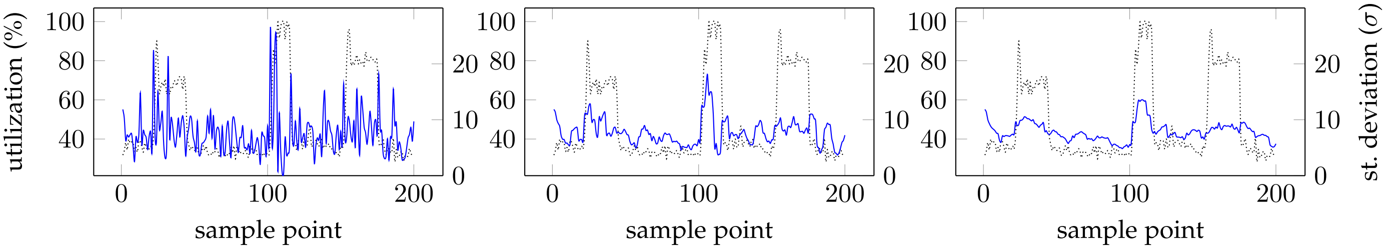


Fig. 5: Standard deviation (σ) of the web server CPU usage signal (black dotted line) for different window sizes (T) (left: $T = 1$, center: $T = 3$, right: $T = 10$).

dynamics can use larger c parameter value with negligible impact on mRT; (iii) abrupt workloads with high frequency dynamics require smaller c parameter value to allow system adaptation to abrupt CPU usage changes.

Hence, we select parameter $c = 0.8$ for future experiments as the workload will be more dynamic and aggressive. Such a large value of parameter c facilitates sufficient resource availability for other applications to run on the same physical machine.

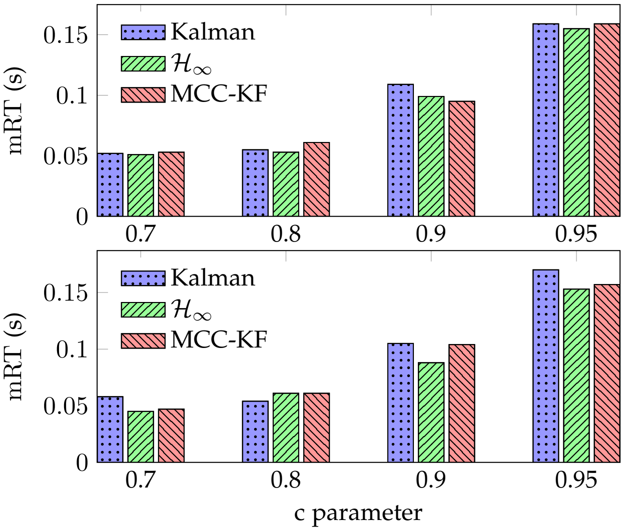


Fig. 6: mRT with respect to c parameter (*Top*: SISO controllers, *bottom*: MIMO controllers). Note mRT increases as c tends to unity.

6.4 Experiment Configuration

The experiments run in total for 250s, an interval adequate enough to evaluate the system's performance. Specifically, two different workload patterns are generated for the experiments, namely Workload 1 (WL1) and Workload 2 (WL2). Both workloads are initiated with 700 clients sending requests to the RUBiS application. At sampling points 10 and 30 another 500 clients are inserted to each workload for about 15 samples. The RUBiS Client Emulator deployed on PM3, sends HTTP requests to the RUBiS application during the entire time of the experiments. The workload type for RUBiS application is set to Browsing Mix (BR), where each client waits for a *think time* [3] following a negative exponential distribution with a mean of 7 seconds (WL1) or a custom think time (WL2) which is included in the default RUBiS workload files, in order to send the next request.

Each experiment that follows (see Fig. 8- 13) has the c parameter set to 0.8. All CPU measurements for the utilization and the allocations are exported from each component through the Xen Hypervisor. The CPU usage measurements are recorded every 1s and after the completion of one sample (i.e., 5s) the mean value of the previous interval is forwarded to the controllers to take action. Using this sampling approach, the control action is applied every 5s in such a way that the high frequency variations of the workload are smoothed and better responses to workload increases are achieved [44]. For the control schemes, the initial value of the error covariance matrix, P_0, the variance of the process noise, W, and the variance of the measurement noise, V are set to 10, 4, and 1 respectively. The values of the process and measurement noises are updated on-line whenever the interval k uses the sliding window approach, mentioned in Section 3.5. The sliding window width T is set to 5 samples, which correspond to 25s.

6.5 Parameter tuning

Both the $\mathcal{H}_\infty$ and MCC-KF require parameter tuning, i.e. θ and σ, respectively. The parameter values are selected via experiments with the workload WL1 described in Section 6.4. Fig. 7 presents the average mRT for the total duration of each experiment with respect to θ and σ value for $\mathcal{H}_\infty$ and MCC-KF (SISO and MIMO models).

Fig. 7 (top histogram) illustrates the different average mRT values when the $\mathcal{H}_\infty$ controller tracks the CPU utilization of each component. For the SISO case, there is a small difference on the mRT for various θ values, however for small values (i.e., 0.1, 0.3) the controller provides extra allocated resources due to under/over estimated gain values. For 0.5 or 0.7 it tracks the CPU utilization accurately. Thus, $\theta = 0.7$ is chosen to maintain CPU utilization tracking as accurate as possible. For the MIMO case, when $\theta = 0.1$ the average mRT is relatively low (0.246s), while for θ being 0.3, 0.5 or 0.7, the average mRT increases due to higher gain (this results to less accurate CPU usage tracking). Thus, to keep CPU usage tracking using the $\mathcal{H}_\infty$ filter non-aggressive (for the experiments) $\theta = 0.1$ is selected.

The bottom histogram of Fig. 7 presents the average mRT for each experiment with respect to σ value for both SISO and MIMO. For the SISO case, when σ equal to 1, 10, 100 and 1000, the average mRT is 0.165s, 0.204s, 0.228s and 0.243s, respectively, while for the MIMO case, the average mRT is 0.171s, 0.155s, 0.256s, and 0.283s, respectively. It is observed that for both cases and for an increasing value of

[3] Time between two sequential requests of an emulated client. Thorough discussion and experiments are presented in [43]

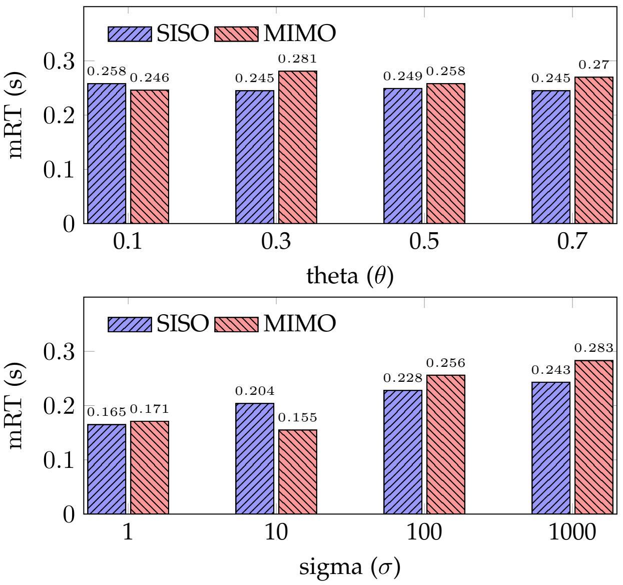


Fig. 7: mRT with respect to tuning parameters (*Top*: θ for $\mathcal{H}_\infty$, and *bottom*: σ for MCC-KF).

σ, the average mRT also increases. This is due to the fact that for low values of σ, the correntropy is dominated by higher-order moments and the estimation does not converge for our setup. According to (5), when $(1+h)\hat{x}_{k+1|k}$ exceeds 100%, the allocation will saturate at the upper bound (i.e., 100%) and, hence, the mRT will remain low due to the abundance of resources. For higher values of σ (e.g., $\sigma = 100$ or $\sigma = 1000$), the usage estimation converges to finite values close to the actual usage. As a result the allocation is maintained (through our allocation mechanism) at levels for which the mRT lies within the SLO. Thus, $\sigma = 100$ seems a good choice for the experiments.

6.6 SISO controllers

Kalman filters are the current state-of-the-art approach for the CPU resource provisioning problem. Fig. 8(a), 8(b) show CPU usages-allocations of both the web server and database server components for the KF. Fig. 8(c) shows the mRT of the RUBiS application requests over time. The KF predicts and adjusts CPU allocations for a duration of 50 samples on both components separately, which facilitates appropriate CPU usage tracking. However, abrupt CPU changes leads to high increase of mRT of the RUBiS application.

The $\mathcal{H}_\infty$ SISO controllers are implemented on both RUBiS application components (see Section 5.3). As $\theta \to 0$ the $\mathcal{H}_\infty$ gain approaches that of the Kalman gain. For our workload profile and experimental testbed, $\theta = 0.7$ for the SISO controller and $c = 0.8$. Fig. 9(a) and 9(b) show the CPU usages-allocations of both the web server and database server components using the $\mathcal{H}_\infty$ filter. Fig. 9(c) presents the mRT of the RUBiS application requests over time. In this experiment, the $\mathcal{H}_\infty$ SISO filter predicts and adjusts the CPU allocations for a duration of 50 samples on both components separately. Evaluation is based on the same workload presented in Section 6.4. Note that, the $\mathcal{H}_\infty$ filter proposed in [29] was evaluated only via simulation using synthetic data. Also, in [41] the $\mathcal{H}_\infty$ filter was applied only on the web server component while the database component was statically fully allocated. Hence, different to the aforementioned works, the $\mathcal{H}_\infty$ filter presented here is evaluated via a real testbed for the first time. Overall, the $\mathcal{H}_\infty$ SISO filter performs well during task of resource allocation since it provides extra resources to the application to maintain low mRT.

Fig. 10(a), 10(b) presents CPU usages-allocations for both web and database servers for the MCC-KF filter, and Fig. 10(c) shows the mRT of the RUBiS application requests. As in the case of the $\mathcal{H}_\infty$ test, the MCC-KF operates well under sudden CPU usage changes and the mRT is low except during periods of workload injection. During all other times, the mRT stays low due to the remaining amount of CPU resources being sufficient for serving a stable workload.

Controller	Kalman	$\mathcal{H}_\infty$	MCC-KF	Workload
Compl. requests	34044	34061	34152	WL1
Avg. VM1 CPU%	62.8	57.6	63.1	
Avg. VM2 CPU%	17.3	17.4	17.4	
Avg. mRT (s)	0.260	0.224	0.246	
SLO obedience	89.0%	90.0%	88.9%	
Compl. requests	36020	36561	36498	WL2
Avg. VM1 CPU%	64.1	64.8	65.4	
Avg. VM2 CPU%	18.6	18.8	18.6	
Avg. mRT (s)	0.908	0.789	0.792	
SLO obedience	58.6%	59.6%	59.6%	

TABLE 1: SISO control evaluation (workloads WL1, WL2).

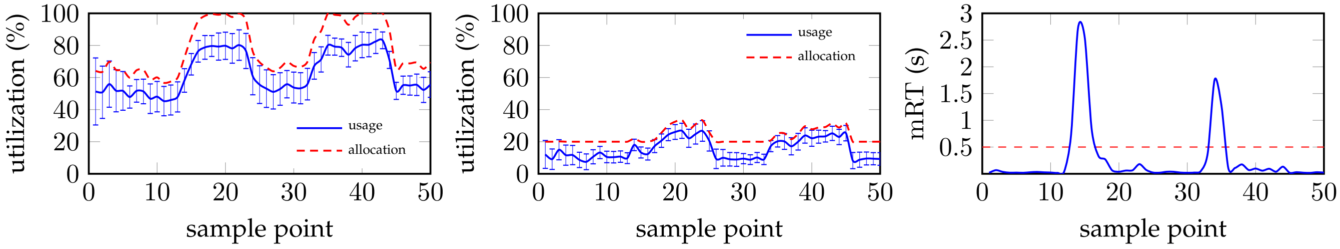
Table 1 presents SISO controller evaluation. For both workloads (WL1 and WL2) the $\mathcal{H}_\infty$ and the MCC-KF perform well (the $\mathcal{H}_\infty$ performing slightly better) and are superior compared to the KF. For example, under WL1 the KF offers an average mRT of 0.260s, while the $\mathcal{H}_\infty$ filter and MCC-KF filter offer average mRT of 0.224s and 0.246s, respectively (similarly for WL2).

6.7 MIMO controllers

Although the Kalman MIMO controller was previously evaluated in [22] and in [24], this work provides a though comparison with the other two MIMO controllers. Fig. 11(a) and 11(b) present CPU usages-allocations for both components during Kalman MIMO controller prediction and CPU resource allocation. Fig. 11(c) shows the RUBiS application mRT using the KF. Fig. 11(a) and 11(b), at sample point 10, shows that the web server utilizations affect directly the database utilizations showing the correlation and inter-component resource couplings between the VMs. An abrupt web server utilization change causes a direct database utilization change, hence affecting mRT. Under stable workload regions the KF optimally allocates resources with unaffected mRT.

Fig. 12(a) and 12(b) present CPU usages-allocations for both components when the $\mathcal{H}_\infty$ MIMO is used. The $\mathcal{H}_\infty$ MIMO controller runs on a remote physical machine in order to predict and control the states of the system for both components to capture the inter-resource couplings. Fig. 12(c) shows RUBiS application mRT over time.

Fig. 13(a), 13(b), and 13(c) present results for the MCC-KF MIMO controller experiment. Kernel size σ of the correntropy criterion is set to 100 to provide sufficient weight in the second and higher-order MCC-KF statistics. The MCC-KF MIMO controller runs on a remote physical machine in order to estimate the states and control the allocation of

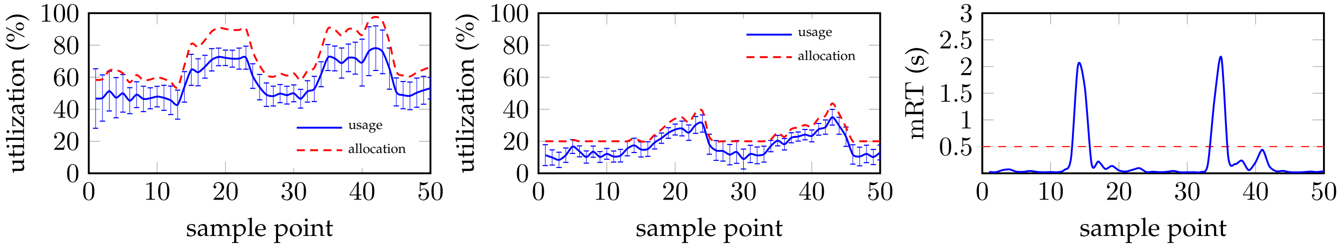


(a)

(b)

(c)

Fig. 8: Kalman - SISO filter. Fig. 8(a): CPU usage and allocation of the web server component. Fig. 8(b): CPU usage and allocation of the database server component. Fig. 8(c): mRT with respect to time for RUBiS application.

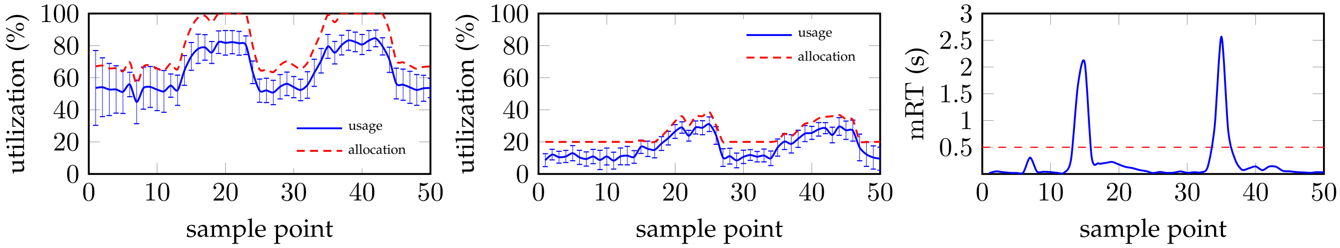


(a)

(b)

(c)

Fig. 9: $\mathcal{H}_\infty$- SISO filter. Fig. 9(a): CPU usage and allocation of the web server component. Fig. 9(b): CPU usage and allocation of the database server component. Fig. 9(c): mRT with respect to time for RUBiS application.



(a)

(b)

(c)

Fig. 10: MCC-KF - SISO filter. Fig. 10(a): CPU usage and allocation of the web server component. Fig. 10(b): CPU usage and allocation of the database server component. Fig. 10(c): mRT with respect to time for RUBiS application.

the system for both components (see Section 5.3). Fig. 13(c) shows the MCC-KF MIMO controller achieved mRT.

Controller	Kalman	$\mathcal{H}_\infty$	MCC-KF	Workload
Cmpl. requests	34053	34402	34332	WL1
Avg. VM1 CPU%	58.4	61.3	61.1	
Avg. VM2 CPU%	17.6	18.0	17.8	
Avg. mRT (s)	0.309	0.270	0.290	
SLO obedience	87.2%	88.1%	88.1%	
Compl. requests	36140	36523	36441	WL2
Avg. VM1 CPU%	64.1	59.9	62.0	
Avg. VM2 CPU%	18.4	18.5	18.8	
Avg. mRT (s)	0.896	0.834	0.835	
SLO obedience.	59.2%	61.7%	61.0%	

TABLE 2: MIMO control evaluation (workloads WL1, WL2).

Table 2 assesses MIMO controllers under workloads WL1 and WL2. The trend is similar to that of SISO, in this context the MCC-KF and $\mathcal{H}_\infty$ filters offer better performance (the $\mathcal{H}_\infty$ being slightly better) than the KF. Note that the KF cannot predict accurately the next state of the system in abrupt workload changes giving the lowest SLO obedience and highest average mRT. Referring to Fig. 14, SISO control gives lower average mRT and higher SLO obedience. $\mathcal{H}_\infty$ offers the best performance with the lowest average mRT and highest SLO obedience. MCC-KF control follows with a slightly higher average mRT. The KF offers the worst performance with the highest average mRT and lowest total SLO obedience.

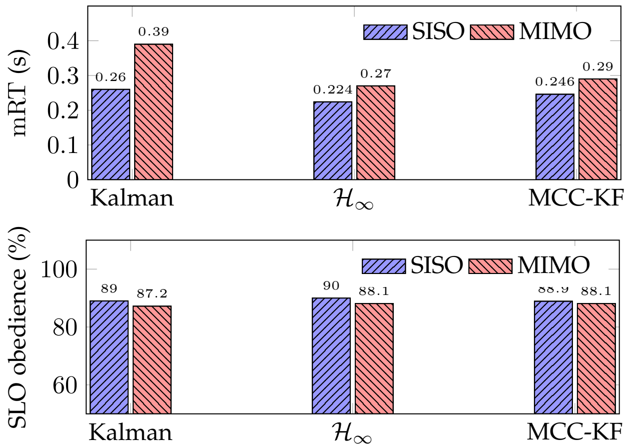
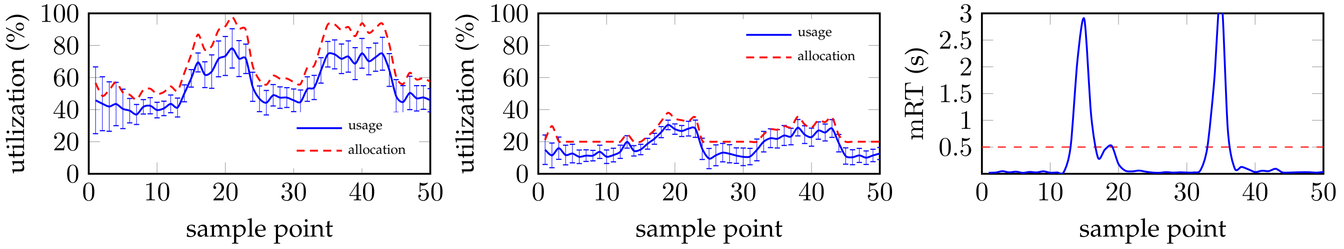
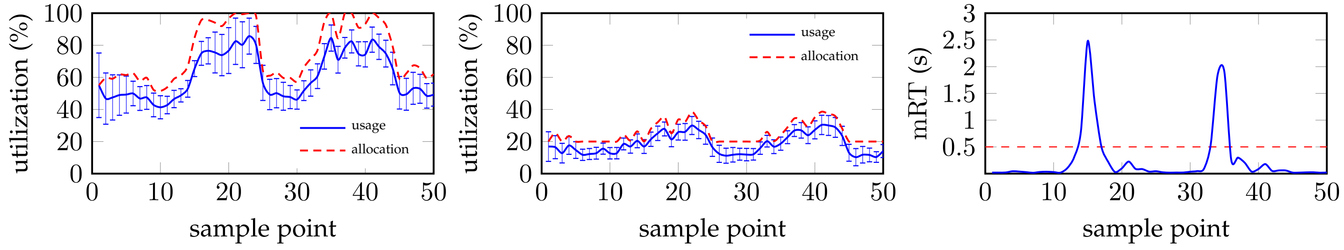


Fig. 14: *Top*: mRT; *bottom*: SLO obedience.



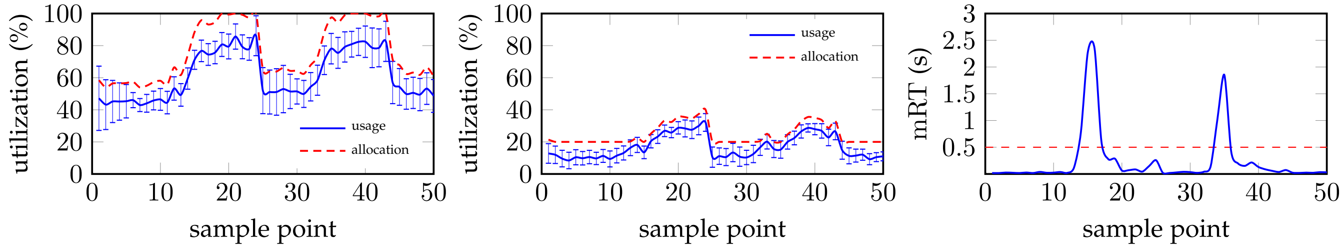
(a) (b) (c)

Fig. 11: Kalman - MIMO filter. Fig. 11(a): CPU usage and allocation of the web server component. Fig. 11(b): CPU usage and allocation of the database server component. Fig. 11(c): mRT with respect to time for RUBiS application.



(a) (b) (c)

Fig. 12: $\mathcal{H}_\infty$- MIMO filter. Fig. 12(a): CPU usage and allocation of the web server component. Fig. 12(b): CPU usage and allocation of the database server component. Fig. 12(c): mRT with respect to time for RUBiS application.



(a) (b) (c)

Fig. 13: MCC-KF - MIMO filter. Fig. 13(a): CPU usage and allocation of the web server component. Fig. 13(b): CPU usage and allocation of the database server component. Fig. 13(c): mRT with respect to time for RUBiS application.

6.8 Statistical Analysis

Here the Student's *t-test* is used to investigate performance improvement of the proposed solutions compared to the current state-of-the-art (KF). In this work, the sample sets contain average mRT values captured during abrupt workload changes obtained from experiments using similar configuration as explained in Section 6.4 for the MIMO case.

Group	Parameter	Value
Kalman filter - $\mathcal{H}_\infty$	*t-statistic* (t) degrees of freedom (df) *p-value* (p)	5.8101 98 7.7669×10^{-8}
Kalman filter - MCC-KF	*t-statistic* (t) degrees of freedom (df) *p-value* (p)	4.3067 98 3.9065×10^{-5}
$\mathcal{H}_\infty$- MCC-KF	*t-statistic* (t) degrees of freedom (df) *p-value* (p)	-1.7481 98 0.0836

TABLE 3: Student's *t-test* results.

Frequent abrupt workload changes were employed to build enough samples for analysis (50 samples). The sample sets (i.e., X_1, X_2 and X_3 for the KF, the $\mathcal{H}_\infty$ and the MCC-KF respectively), are independent samples from normal distributions as each experiment does not affect the others. To assess significant differences between two samples, we firstly define two hypotheses: (a) Null Hypothesis (H_0) - no significant difference between the two samples; (b) Alternative Hypothesis (H_a) - significant difference between the two samples. The degrees of freedom, *t-statistic*, and the *p-value* follow in Table 3.

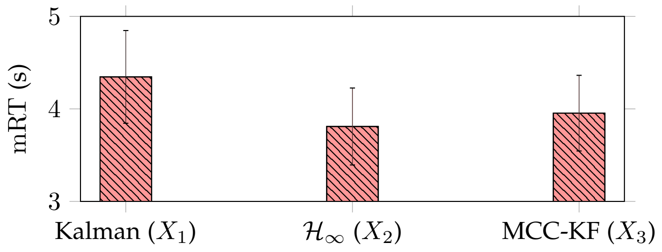


Fig. 15: Statistical results. Means: $\mu_{X_1} = 4.346$, $\mu_{X_2} = 3.810$, $\mu_{X_3} = 3.954$; standard deviation: $\sigma_{X_1} = 0.501$, $\sigma_{X_2} = 0.416$, $\sigma_{X_3} = 0.409$.

A significance level of $\alpha = 0.05$ is assumed, and the null hypothesis (H_0) rejected for both groups: KF-$\mathcal{H}_\infty$, and KF-

(MCC-KF) due to $p < \alpha$. However, the null hypothesis is not rejected for group $\mathcal{H}_\infty$-(MCC-KF) since $p > \alpha$. Thus, significant difference between the two proposed filters and the KF applies, while for the $\mathcal{H}_\infty$-(MCC-KF) group does not apply. From Fig. 15 - which shows mean and standard deviation of mRT samples - and from the *t-test* results, one concludes that the $\mathcal{H}_\infty$ and MCC-KF perform significantly better (i.e., lower average mRT during abrupt workload changes) compared to the KF.

7 Remarks

In addition, the following remarks are highlighted:

1) Robust state estimation performs well in predicting abrupt workload changes even when the process noise is non-Gaussian which is usually the case in practical implementations (cloud applications demand).
2) SISO controllers consider no inter-component resource couplings due to their independent action on each physical machine. Also, no delays are experienced as no message exchanging with other physical machines applies.
3) SISO control enables each cloud application component to host a different controller based on workload variation (e.g. KF for smooth CPU usage components, $\mathcal{H}_\infty$ or MCC-KF for abrupt ones).
4) MIMO controllers encompass inter-component resource couplings via the covariance calculation process.
5) MIMO controllers are installed on remote servers which may experience network delays that can hinder application performance. However, as modern cloud applications are hosted on multi-tier applications, MIMO control is preferable to centrally estimate the allocations of each component while considering resource couplings.

8 Conclusions and Future Directions

This paper presented a rigorous study of SISO and MIMO models comprising adaptive robust controllers for the CPU resource allocation problem of VMs and satisfying certain QoS requirements. The controllers aimed to adjust the CPU resources based on observations of previous CPU utilizations. Tests were performed on a two-tier cloud application experimental platform. The proposed controllers offer improved performance under abrupt and random workload changes compared to the current state-of-the-art. Results show that (a) SISO controllers perform better than MIMO; (b) $\mathcal{H}_\infty$ and MCC-KF offer improved performance than the KF for abrupt workloads; (c) $\mathcal{H}_\infty$ and MCC-KF have similar performance for both SISO and MIMO models. The proposed robust controllers successfully reduce average mRT while keeping SLO violations low.

The system in this work addresses only CPU capacity, and resource needs are coupled across multiple dimensions (i.e., compute, storage, and network bandwidth). Hence, workload consolidation is required while catering for resource coupling in multi-tier virtualized applications to provide timely allocations for abrupt workload changes. Ongoing research investigates system identification/learning to extract coupling information between resource needs for workload consolidation while meeting SLOs.

References

[1] E. R. Masanet, R. E. Brown, A. Shehabi, J. G. Koomey, and B. Nordman, "Estimating the Energy Use and Efficiency Potential of U.S. Data Centers," *Proc. of the IEEE*, vol. 99, no. 8, pp. 1440–1453, Aug 2011.

[2] S. Bhowmik, *Cloud Computing*. Cambridge University Press, 2017.

[3] T. H. Nguyen, M. Di Francesco, and A. Yla-Jaaski, "Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers," *IEEE Trans. on Services Comp.*, 2017.

[4] A. Ullah, J. Li, Y. Shen, and A. Hussain, "A control theoretical view of cloud elasticity: taxonomy, survey and challenges," *Cluster Computing*, May 2018.

[5] J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 64, pp. 23–42, 2016.

[6] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: state of the art and research challenges," *IEEE Trans. on Services Computing*, vol. 11, no. 2, pp. 430–447, 2018.

[7] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-Theoretical Software Adaptation: A Systematic Literature Review," *IEEE Transactions on Software Engineering*, vol. 44, no. 8, pp. 784–810, Aug 2018.

[8] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Software engineering meets control theory," in *Proc. of the Int'l Symp. on Softw. Eng. for Adaptive and Self-Managing Syst.*, ser. SEAMS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 71–82.

[9] A. Filieri, M. Maggio, K. Angelopoulos, N. D'ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Control strategies for self-adaptive software systems," *ACM Trans. Autonom. Adapt. Syst.*, vol. 11, no. 4, pp. 24:1–24:31, Feb. 2017.

[10] X. Zhu, Z. Wang, and S. Singhal, "Utility-Driven Workload Management using Nested Control Design," in *Proc. of the American Control Conf. (ACC)*, 2006, pp. 6033–6038.

[11] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, and S. Singhal, "AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments," in *Proc. of the IEEE Int'l Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, 2007.

[12] 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 *Proc. of the Europ. Conf. on Computer Syst. (EuroSys)*, 2007, pp. 289–302.

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

[14] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Controlling software applications via resource allocation within the heartbeats framework," in *49th IEEE Conf. on Decision and Control (CDC)*. IEEE, 2010, pp. 3736–3741.

[15] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Trans. on Services Computing*, vol. 4, no. 5, pp. 497–511, 2012.

[16] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications," in *Proceedings of the 2016 24th ACM SIGSOFT Int'l Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 217–228.

[17] E. B. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth, "Towards faster response time models for vertical elasticity," in *2014 IEEE/ACM 7th Int'l Conf. on Utility and Cloud Computing*. IEEE, 2014, pp. 560–565.

[18] S. Spinner, S. Kounev, X. Zhu, L. Lu, M. Uysal, A. Holler, and R. Griffith, "Runtime vertical scaling of virtualized applications via online model estimation," in *2014 IEEE 8th Int'l Conf. on Self-Adaptive and Self-Organizing Systems*. IEEE, 2014, pp. 157–166.

[19] L. Yazdanov and C. Fetzer, "Vertical scaling for prioritized vms provisioning," in *2012 Second Int'l Conf. on Cloud and Green Computing*. IEEE, 2012, pp. 118–125.

[20] P. A. Dinda and D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Computing*, vol. 3, no. 4, pp. 265–280, 2000.

[21] E. Kalyvianaki, T. Charalambous, and S. Hand, "Resource Provisioning for Multi-Tier Virtualized Server Applications," *Computer Measurement Group (CMG) Journal*, vol. 126, pp. 6–17, 2010.

[22] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers using Kalman Filters," in *Proc. of the 6th Int'l Conf. on Autonom. Comput. (ICAC)*. NY, USA: ACM, 2009, pp. 117–126.

[23] 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 4th ACM Europ. Conf. on Comp. Syst. (EuroSys '09)*. NY, USA: ACM, 2009, pp. 13–26.

[24] E. Kalyvianaki, T. Charalambous, and S. Hand, "Adaptive resource provisioning for virtualized servers using Kalman filters," *ACM Trans. on Autonom. and Adapt. Syst.*, vol. 9, no. 2, pp. 10:1–10:35, July 2014.

[25] E. B. Lakew, A. V. Papadopoulos, M. Maggio, C. Klein, and E. Elmroth, "Kpi-agnostic control for fine-grained vertical elasticity," in *Proceedings of the 17th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 589–598.

[26] D. C. Marinescu, *Cloud Computing: Theory and Practice*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[27] S. Shevtsov, D. Weyns, and M. Maggio, *Self-Adaptation of Software Using Automatically Generated Control-Theoretical Solutions*. Singapore: Springer Singapore, 2019, pp. 35–55.

[28] L. Kleinrock, *Queueing Systems, Volume 1, Theory*. Wiley-Interscience, 1975.

[29] T. Charalambous and E. Kalyviannaki, "A min-max framework for CPU resource provisioning in virtualized servers using $\mathcal{H}_\infty$ Filters," in *IEEE Conf. on Decision and Control (CDC)*, Dec. 2010, pp. 3778–3783.

[30] K. M. Deliparaschos, T. Charalambous, E. Kalyvianaki, and C. Makarounas, "On the use of fuzzy logic controllers to comply with virtualized application demands in the cloud," in *European Control Conf. (ECC)*, June 2016, pp. 649–654.

[31] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *2011 31st Int'l Conf. on Distributed Computing Systems*. IEEE, 2011, pp. 571–580.

[32] R. N. Banavar, "A game theoretic approach to linear dynamic estimation," Ph.D. dissertation, Texas Univ., Austin, July 1992.

[33] D. Simon, *Optimal State Estimation: Kalman, H-infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[34] H. Poveda, E. Grivef, G. Ferré, and N. Christov, "Kalman vs $\mathcal{H}_\infty$ filter in terms of convergence and accuracy: Application to carrier frequency offset estimation," in *Proc. of the 20th European Signal Processing Conf. (EUSIPCO)*, Aug 2012, pp. 121–125.

[35] B. Chen, X. Liu, H. Zhao, and J. C. Principe, "Maximum correntropy Kalman filter," *Automatica*, vol. 76, pp. 70–77, 2017.

[36] R. Izanloo, S. A. Fakoorian, H. S. Yazdi, and D. Simon, "Kalman filtering based on the maximum correntropy criterion in the presence of non-Gaussian noise," in *Annual Conf. on Information Science and Systems (CISS)*, Mar. 2016, pp. 500–505.

[37] W. Liu, P. P. Pokharel, and J. C. Principe, "Correntropy: A localized similarity measure," in *Neural Networks, 2006. IJCNN'06. Int'l Joint Conf. on*. IEEE, 2006, pp. 4919–4924.

[38] W. Liu, P. P. Pokharel, and J. C. Príncipe, "Correntropy: Properties and Applications in Non-Gaussian Signal Processing," *IEEE Trans. on Signal Processing*, vol. 55, no. 11, Nov 2007.

[39] R. He, W. S. Zheng, and B. G. Hu, "Maximum Correntropy Criterion for Robust Face Recognition," *IEEE Trans. on Pattern Anal. and Machine Intell.*, vol. 33, no. 8, pp. 1561–1576, Aug. 2011.

[40] A. Singh and J. C. Principe, "Using Correntropy As a Cost Function in Linear Adaptive Filters," in *Proc. of the 2009 Int'l Joint Conf. on Neural Networks*, ser. IJCNN'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1699–1704.

[41] E. Makridis, K. Deliparaschos, E. Kalyvianaki, and T. Charalambous, "Dynamic CPU Resource Provisioning in Virtualized Servers using Maximum Correntropy Criterion Kalman Filters," in *IEEE Int'l Conf. on Emerg. Tech. and Factory Autom. (ETFA)*, Sept. 2017.

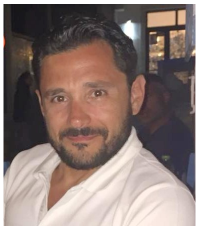
[42] C. Amza, A. Chandra, A. L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and Implementation of Dynamic Web Site Benchmarks," in *Proc. of the 5th Ann. IEEE Int'l Workshop on Workload Charact. (WWC-5)*, 2002, pp. 3–13.

[43] A. Bahga and V. K. Madisetti, "Synthetic workload generation for cloud computing applications," *Journal of Software Engineering and Applications*, vol. 4, no. 07, p. 396, 2011.

[44] E. Kalyvianaki, "Resource provisioning for virtualized server applications," Univ. of Cambridge, Computer Lab., Tech. Rep., 2009.

Evagoras Makridis received his BSc degree in Electrical Engineering from Cyprus Univ. of Technology, Cyprus. Since September 2018, he is a MSc student in the Autonomous Systems program provided by EIT Digital Master School at the Dept. of Electr. Eng. and Autom., School of Electr. Eng., Aalto Univ.

His research interests focus on cloud control, autonomous and controlled systems.



Kyriakos Deliparaschos, BEng Electron. Eng. from De Montfort Univ. (DMU), MSc Mechatronics from DMU and Nat. Tech. Univ. of Athens (NTUA), PhD from NTUA. He is special teaching staff at EECEI Dept., Cyprus Univ. of Technol. (CUT), and also Res. Assoc. (RA) at NTUA's IRA Lab. He was Postdoc. Res. Fellow with CTVR group, Trinity College Dublin and RA with the RCDS Lab., MEMSE Dept. of CUT. His research interests are in high performance computing, embedded and cyber-physical systems, HW accelerators, mobile robots and robot-assisted surgery, cloud computing and fault tolerance control.

Evangelia Kalyvianaki is a Senior Lecturer (Assistant/Associate Professor) in the Dept. of Comp. Sci. and Tech. at Univ. of Cambridge and member of the SRG/netos group. Before, she was a Lecturer at the Dept. of Comp. Sci. at City Univ. London and a Postdoctoral researcher in the Dept. of Comp., Imperial College London. She obtained her PhD from the Comp. Lab. (SRG/netos group) in Cambridge Univ. She holds an MSc and a BSc degrees from the Comp. Sci. Dept. of the Univ. of Crete, Greece.

Her research interests span the areas of cloud computing, big data processing, autonomic computing, and systems research in general.

Argyrios Zolotas (SM'11), Univ. of Leeds (B.Eng Hons Class I), Loughborough Univ. (PhD), Univ. of Leicester (MSc, Distinction). He is Reader at Cranfield Univ., he was with Univ. of Lincoln, Univ. of Sussex, Loughborough Univ., and Imperial College London. He was visiting Prof. at Grenoble INP in May-June 2018. His research interests are in advanced control, systems autonomy, digital engineering.

Themistoklis Charalambous received his BA and MEng in Electrical and Information Sciences from the Univ. of Cambridge. He completed his PhD studies in the Control Lab., Univ. of Cambridge. He worked as Research Associate at Imperial College London, as a Visiting Lecturer at the Dept. of Electr. and Comp. Eng., Univ. of Cyprus, as a Postdoctoral Researcher at the Dept. of Autom. Control of the School of Electr. Eng. at the Royal Institute of Technology (KTH) and the Dept. of Electr. Eng. at Chalmers Univ. of Tech. Since 2017, he is an Assistant Professor at the School of Electr. Eng., Aalto Univ. His primary research targets the design and analysis of networked control systems that are stable, scalable and energy efficient.

2020-01-15

Robust dynamic CPU resource provisioning in virtualized servers

Makridis, Evagoras

IEEE