Adaptive Resource Allocation for Computation Offloading: A Control-Theoretic Approach

MARIOS AVGERIS and DIMITRIOS DECHOUNIOTIS, National Technical University of Athens
NIKOLAOS ATHANASOPOULOS, Queen's University Belfast, Northern Ireland
SYMEON PAPAVASSILIOU, National Technical University of Athens

Although mobile devices today have powerful hardware and networking capabilities, they fall short when it comes to executing compute-intensive applications. Computation offloading (i.e., delegating resource-consuming tasks to servers located at the edge of the network) contributes toward moving to a mobile cloud computing paradigm. In this work, a two-level resource allocation and admission control mechanism for a cluster of edge servers offers an alternative choice to mobile users for executing their tasks. At the lower level, the behavior of edge servers is modeled by a set of linear systems, and linear controllers are designed to meet the system's constraints and quality of service metrics, whereas at the upper level, an optimizer tackles the problems of load balancing and application placement toward the maximization of the number the offloaded requests. The evaluation illustrates the effectiveness of the proposed offloading mechanism regarding the performance indicators, such as application average response time, and the optimal utilization of the computational resources of edge servers.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → *Computational control theory*; • **Human-centered computing** → Ubiquitous and mobile computing design and evaluation methods;

Additional Key Words and Phrases: Edge computing, linear modeling, feedback control

ACM Reference format:
Marios Avgeris, Dimitrios Dechouniotis, Nikolaos Athanasopoulos, and Symeon Papavassiliou. 2019. Adaptive Resource Allocation for Computation Offloading: A Control-Theoretic Approach. *ACM Trans. Internet Technol.* 19, 2, Article 23 (April 2019), 20 pages.
https://doi.org/10.1145/3284553

1 INTRODUCTION

Over the past decades, the processing and networking capabilities of mobile devices have grown significantly. This has allowed for the development of mobile applications for a wide range of human daily activities, including healthcare and wellness, education, commerce, and social

Authors' addresses: M. Avgeris, D. Dechouniotis, and S. Papavassiliou, National Technical University of Athens, 9, Iroon Polytechniou Street, Zografou, 157 80, Greece; emails: {mavgeris, ddechou}@netmode.ntua.gr, papavass@mail.ntua.gr; N. Athanasopoulos, Queen's University Belfast, Ashby Building, Stranmillis Road, BT9 5AG, Belfast, Northern Ireland, UK; email: n.athanasopoulos@qub.ac.uk.

1533-5399/2019/04-ART23 $15.00
https://doi.org/10.1145/3284553

media. However, the constrained computing resources and battery capacity of mobile devices still remain an obstacle for the realization of compute-intensive and high energy consuming applications. Mobile cloud computing (MCC) is the emerging service delivery paradigm that integrates cloud computing (CC) into the mobile environment. MCC provides on-demand, low-latency, and secure access to a resourceful group of servers in the spatial vicinity of mobile users. This comes complementary to the CC paradigm, which suffers from latency issues due to the connection to remote servers in the cloud through public Internet. MCC includes slightly different architectures, such as edge computing, mobile edge computing, and fog computing [20]. The common characteristic that these architectures share is the placement of a cluster of servers at the edge of the network. For the rest of the article, these servers are called *edge servers* independently of the architecture. Edge servers receive and execute compute-intensive tasks of mobile applications. The problem of determining what task, where, and whether it should be offloaded to save energy and/or meet time constraints is known as *computation offloading* [16].

Currently and regardless of the adopted MCC architecture, the computation offloading decision is coupled with the resource allocation in the edge servers. In contrast to the CC setting, a cluster of edge servers does not have abundant resources. Consequently, together with the offloading decision, a dynamic resource allocation and admission control mechanism, which we call *vertical and horizontal scaling*, is necessary. This mechanism is responsible for the (de)activation of the edge servers, the placement of the application instances, and the distribution of offloaded requests among them, which we call *horizontal scaling*. Alongside is the admission control and resource allocation for each application instance within the active servers, which we call *vertical scaling*. As it is referred in Section 2, most of the proposed studies in the literature use queuing theory to model mobile devices and edge servers, along with an optimization method for finding the optimal offloading policy. The most commonly used criteria are the energy consumption of the mobile device and the request throughput. There are two major shortcomings of these approaches that can lead to the deterioration of the system's performance. First, the static modeling of the servers for fluctuating workload can lead to overprovisioning or underprovisioning. Second, there is no formal guarantee of satisfying the physical constraints—for instance, CPU and memory sharing, or meeting the quality of service (QoS) specifications, such as the average response time. Since all MCC architectures use a small cluster of servers, a fallacious resource allocation mechanism can hamper the offloading performance.

Contrary to the CC environments where dynamic modeling and control mechanisms have been extensively adopted [17, 18], little attention has been given to the optimal use of the edge servers. In this work, we develop a two-level cooperative resource allocation mechanism for a single cluster of edge servers hosting a group of applications, which allows mobile users, within the coverage area, to offload application-specific tasks. It should be noted, however, that user mobility within the cluster's proximity has not been considered in this work and is left for future research. The proposed mechanism can on-demand allocate the edge servers' resources to different applications using virtual machines (VMs). At the lower level, the dynamic operation of VMs is captured by linear dynamics. The local controllers are responsible for regulating allocated CPU shares and accepted offloading requests, according to a varying, but bounded in a given interval, incoming workload. This comprises the vertical scaling part of our mechanism. At the upper level, a horizontal scaling process is responsible for activating the essential number of edge servers and placing the appropriate VMs into them. This comprises the horizontal scaling part. In particular, the incoming requests are distributed to the activated servers to serve the total demand. This process is orchestrated while taking the local controllers into consideration, making this mechanism cooperative. The benefit of this approach is manifold.

At the lower level:

— our modeling approach can accurately capture the dynamic behavior of the application-specific VM under different operating conditions;
— a multitude of feasible operating points can be calculated, considering different performance and utilization costs, which allows us to design different control strategies for different pairs of workload and applications; and
— formal guarantees regarding resource allocation and QoS specifications are provided.

At the upper level:

— the minimum number of edge servers is activated to satisfy the overall workload of all applications, based on the set of the feasible operating points of the lower level.

The rest of the article is organized as follows. Section 2 discusses related work. Section 3 presents the proposed modeling, alongside the vertical and horizontal scaling methodologies on the computation offloading at the edge servers. Detailed performance evaluation and comparison with an energy-aware offloading technique [30] are illustrated in Section 4, and conclusions are drawn and opportunities for future research are identified in Section 5.

2 RELATED WORK

One of the initial and influential works on MCC [25] proposed a dynamic VM synthesis of a cloudlet infrastructure. The position paper of Satyanarayanan et al. [26] presented the potentials of MCC ecosystems: wearable devices, Internet of Things (IoT) applications, and automotive and industrial environments alongside tactile Internet can leverage from the mobile-cloud convergence. The extended survey of Sanaei et al. [24] presented a definition of MCC, the vision and the challenges, and a taxonomy of heterogeneity in MCC and open issues. The survey paper of Mahmud et al. [20] analyzed the challenges of fog computing in terms of architecture, service, and security, and classified the existing studies according to these criteria.

Surveys of existing computation offloading approaches are provided by Kumar et al. [16] and Bhattacharya and De [2]. Huang et al. [10] addressed computation offloading as an admission control problem in MCC hotspots with a cloudlet, using semi-Markov decision process modeling and linear programming. The resource constraints were considered when obtaining the optimal solution. A similar dynamic offloading algorithm was proposed by Zhang et al. [32]. Therein, the admission control problem on cloudlets was modeled and solved as a Markov decision process, aiming to minimize the computation and offloading costs. The mobility of the users was taken into account as well. Khojasteh et al. [13] presented two flexible resource allocation algorithms for computation offloading. The resource allocation process and VM provisioning were modeled by a Markovian multiserver queuing system with priority levels and a multidimensional Markov system, based on a birth-death queuing system with finite population, respectively. In the work of Raei and Yazdani [22], three resource allocation schemes were proposed for computation offloading. Several stochastic sub-models captured the operation of a physical machine, under the policy of each scheme. The Markov reward model was applied to obtain the output of the sub-models, and the decision criteria consist of the request rejection probability and mean response delay. Cardellini et al. [5] proposed a hierarchical MCC architecture where users could offload their tasks, modeled by queuing models, either to local cloudlets or the remote public cloud. Computation offloading was modeled as a generalized Nash equilibrium problem, and a distributed algorithm computed an equilibrium strategy for each user.

Many studies focus on energy-aware offloading. In the work of Xia et al. [30], a two-tier MCC environment was adopted; mobile devices, cloudlets, and the remote cloud were described by static models, and an algorithm that optimized the minimum residual energy ratio was developed. Jalali et al. [11] proposed static, flow-based, and time-based energy consumption models. They presented a detailed energy consumption comparison between CC and fog computing architectures while taking the network equipment into account. Their numerical results demonstrated how offloading leads to energy saving for IoT applications. In the work of Kiani and Ansari [14], a task scheduling scheme for code partitioning in a hierarchical cloudlet environment was proposed for two different use cases. The one finds the optimal task scheduling for already defined radio parameters, whereas the other optimizes both the task scheduling and the transmission power of the mobile devices.

Finally there are some interesting studies that examine other problems in the area of computation offloading. In the work of Barbera et al. [1], the feasibility of computation offloading and data backups in real-life scenarios was examined. Since communication is not free, the authors focused on bandwidth and power consumption of WiFi, 2G, and 3G technologies. A real testbed with smartphones and Amazon EC2 nodes was used for thorough analysis. Xu et al. [31] focused on cloudlet placement to minimize the average cloudlet access delay between mobile devices and cloudlets. A heuristic scalable algorithm was proposed for the special case of homogeneous cloudlets. Jia et al. [12] used the placement of Xu et al. [31] and proposed a load balancing algorithm to utilize fairly a group of cloudlets. Queuing models were adopted for cloudlets and a scalable algorithm computed the optimal request redirection such that the maximum of the average response times at cloudlets was minimized. Trying a different approach, Liu and Liu [19] proposed a game-based distributed MEC scheme where the users competed for the cloudlet's finite computation resources via a pricing approach, modeled as a Stackelberg game. The algorithms examined there were implemented in a distributed manner. Sun and Ansari [28] proposed two algorithms for maintaining the low end-to-end delay between the mobile devices and the cloudlets when the users move around the network topology. The key idea lies in optimally deploying the mobile device's corresponding VMs in the available cloudlets while adapting to the user's movement. Dealing with the opposite dataflow (i.e., offloading from the cloud to the edge), Cao and Papadimitriou [4] presented a collaborative content caching system at the network edge. They developed a model to instruct the edge node to trigger on demand caching when popular content has been identified. SDN techniques were leveraged to manage and distribute the content among the access nodes in a coordinated manner.

A shortcoming of some studies [1, 11, 12, 30, 31] is that the modeling of the edge servers captures accurately a single operating point and not the whole operating range. However, for the various Markov process approaches [10, 13, 22, 32], the execution time of each request was derived from a fixed service rate. However, these assumptions on the static operating range and service rate apply only when the operating conditions are close to that point. Furthermore, in the preceding studies, a systematic analysis on satisfaction of the QoS specifications and the constraints is missing. The present study aims to address the aforementioned shortcomings. Thus, state-space modeling is used to capture the dynamic behavior of the edge server under different operating conditions. The local controller computes the system's feasible operating (equilibrium) points while considering different competitive criteria, and guarantees the stability and confinement in a specific area around them. The horizontal scaler takes these operating points into account and determines the appropriate placement that serves the incoming varying workload.

3 COMPUTATION OFFLOADING

Computation offloading mitigates the energy consumption of resource-constrained mobile devices by relocating the execution of the compute-intensive tasks to a group of edge servers that are placed in the mobile users' spatial vicinity. This placement enables low-latency access to the servers, contrary to the access to the remote cloud through the public Internet, which is unpredictable when it comes to response times. Figure 1 depicts the MCC computation offloading

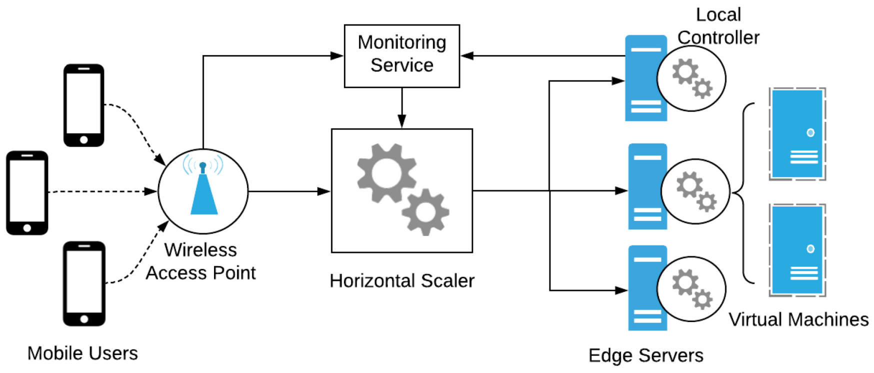


Fig. 1. MCC computation offloading architecture.

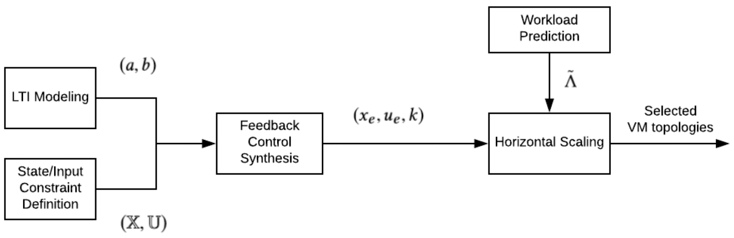


Fig. 2. MCC computation offloading mechanism worklflow.

architecture studied in this article. Specifically, the offloaded traffic, generated from the mobile devices, is directed to the horizontal scaler through the local wireless access point (with WiFi, 3G/4G, or LTE support). There lies the upper-level control process of our mechanism; this component selects an appropriate VM placement to be implemented to each edge server directly connected to it and consequently distributes the incoming workload accordingly. This decision defines the number of active servers alongside the number and the operating state of the VMs to be placed in them. This upper-level process is performed in an online and proactive manner through the use of an internal prediction mechanism, the workload predictor, described in more detail in Section 3.3.2, able to estimate the incoming offloading requests in the following time window. The essential input for this estimation process is provided by the monitoring service component, which is responsible for collecting data regarding both the network traffic (e.g., offloading requests issued, end-to-end response times) and the servers' resources utilization (e.g., CPU usage) at each given time. As mentioned earlier, this is the horizontal scaling part of our mechanism, and the theory behind it is described thoroughly in Section 3.3.

At the lower level, each edge server is equipped with a local controller, which is able to create, run, scale, and stop application-specific VMs, thus assisting the realization of the selected VM placement for the given time window. Additionally, the lower-level control process is implemented in this component, as it moderately scales the VMs vertically based on data coming from the monitoring service. In this way, it ensures that the VMs remain within the selected operating state, thus guaranteeing minimum and stable application response times. The theoretical design behind this control process is described in more detail in Section 3.2.

Figure 2 illustrates the workflow of the proposed MCC computation offloading mechanism. In the proposed approach, the operation of the VMs is modeled by a group of linear time invariant (LTI) systems that are subject to additive exogenous disturbances. The parameters a, b of the LTI

Table 1. Table of Notations

t	Time instant
λ	Incoming RR per second
x	Average response time (seconds)
u	Input $u(t) = [u_1(t) \quad u_2(t)]^\top$; u_1 is the allocated VM CPU; u_2 is the RR admitted at the VM
(x_e, u_e)	Feasible operating point
r_e	Statically allocated VM memory
w	Communication disturbances
a, b	LTI system parameters
k	Control gain
$\mathbb{X}, \mathbb{U}$	State and input constraints
$\mathcal{S}$	Invariant set
$\mathcal{P}$	Set of feasible VM placements
E	Number of edge servers
$\tilde{\Lambda}$	Predicted incoming RR

systems in (1) are identified by experimental data. At first, for each LTI system, a feasible equilibrium of the nominal disturbance-free model of the VM, (x_e, u_e), is computed. Each equilibrium point corresponds to an operating state of the VM without assuming disturbances. For example, an operating point might correspond to three requests per second, utilizing 20% of CPU allocation and resulting in an average response time of 3 seconds For each equilibrium point, a linear state feedback controller, meaning the control gain k, is designed by taking the disturbed system into account, within the local controller component. Specifically, by regulating the assigned CPU allocation and the number of admitted requests, we design a controller such that the closed loop system (i) is stable, (ii) satisfies the constraints and the QoS specifications at all times and for any initial condition, starting from within the constraint set, and (iii) behaves optimally in steady state. Since the proposed resource management mechanism offers guaranteed response time to mobiles users, the offloading decision breaks down to a simple comparison between the estimated execution time on the mobile device and the guaranteed response time provided by the edge servers.

At the upper level, for each application, the horizontal scaler receives an estimation of the forthcoming requests $\tilde{\Lambda}$, made by the workload predictor component, and the set of the feasible operating points (x_e, u_e), computed by the feedback controller, as input. Then, based on this information, it decides the minimum number of active edge servers and the VM placement to be implemented in them, toward the satisfaction of the total demand for each application. This cooperation of the two control levels ensures that the selected operating point of each VM from the horizontal scaler will be realized by the feedback controller of the VM. The following table sums up the main symbols used in the next sections, alongside their description.

3.1 System Modeling

In our setting, a number of $N \geq 1$ different applications are hosted in isolated VMs in an edge server. For each application and for a range of incoming RRs, a scalar discrete LTI system is identified. To this purpose, let

$$\lambda(t) \in [\Lambda_m, \Lambda_M]$$

denote the incoming request rate (RR) per second at time instant t, which is varying in an interval $[\Lambda_m, \Lambda_M]$. The range of incoming RR is divided in L subintervals of the form $[\Lambda_{i,m}, \Lambda_{i,M}] \subset [\Lambda_m, \Lambda_M]$, $i = 1, \ldots, L$. Consequently, for each application and each RR subinterval, a linear system with additive disturbances is identified of the form

$$x(t+1) = \max\{ax(t) + bu(t) + w(t), 0\}. \tag{1}$$

In the preceding equation, $x(t)$ is the average response time, $u(t) = [u_1(t) \quad u_2(t)]^\top \in \mathbb{R}^2$ is the input vector, and $w(t) \in [w_m, w_M]$, $w_m < 0 < w_M$, is an unknown, however bounded, signal, which accounts for the disturbances induced by the communication between the edge server and the mobile users and the modeling error of the identified model. In our case, we focus on CPU-intensive applications, and thus VM memory is statically assigned and not included in the linear systems. However, the horizontal scaler takes memory constraints in the VM placement problem into account, as described in Section 3.3.1. To simplify notation, we do not indicate that the exposition is done for system (i), since this is arbitrarily chosen.

The input $u_1 \in [u_{1m}, u_{1M}]$ corresponds to the allocated CPU share of the VM, whereas $u_2 \in [\Lambda_{i,m}, \Lambda_{i,M}]$ is the RR the controller admits. The parameter $a \geq 0$ is a known scalar, whereas $b \in \mathbb{R}^{1\times 2}$ is a row vector. Both a, b can be estimated using the recursive least square (RLS) algorithm [29]. The maximum operator in (1) ensures that there are no negative average response times in the model.

The state and input variables x and u are inherently constrained due to the finite resources and the control specifications. Specifically,

$$\mathbb{X} := \{x : \ 0 \leq x(t) \leq X_M\}, \tag{2}$$

$$\mathbb{U} := \{u : \ u_m \leq u(t) \leq u_M\}, \tag{3}$$

for all $t \geq 0$, where $u_m = [u_{1m} \quad \Lambda_{i,m}]^\top$, $u_M = [u_{1M} \quad \Lambda_{i,M}]^\top$.

By having a set of models (1) corresponding to different RR intervals, we provide better level of accuracy than a single LTI model for the whole range of RRs. Additionally, the number of models scales linearly with respect to L, since we consider the co-hosted applications are decoupled and depend only on the number of subintervals of the RR.

At the lower level, the local controllers focus on the joint resource allocation and admission control of edge servers to perform guaranteed response time under the varying workload of the consolidated applications. For the lower control level (vertical scaling), our goals are summarized as follows:

— ***P1*:** Consider system (1) subject to the constraints (2), (3) that corresponds to a certain incoming RR. Given a desired response time, find a *feasible* operating region for the system (1) that is optimal with respect to a well-defined cost.

— ***P2*:** For each operating condition calculated in **P1**, compute an admissible control strategy that steers the closed loop system to it and respects the constraints at all times.

3.2 Vertical Scaling

In this section, we discuss how our approach tackles the problems P1, P2 *simultaneously*. Specifically, an optimization problem is formulated whose solution retrieves both the operating condition and the control strategy. This approach is less conservative than the multistep approaches in the literature [7, 8].

Let us consider an admissible equilibrium pair (x_e, u_e) for the disturbance-free system (1)—for instance, when $w(t) = 0$, for all $t \geq 0$. Clearly, x_e and u_e satisfy the equation

$$x_e = ax_e + bu_e$$

and satisfy the constraints (2), (3). An affine state feedback control laws of the form is considered:

$$u(t) = k(x(t) - x_e) + u_e, \quad t \geq 0, \tag{4}$$

where $k \in \mathbb{R}^2$ is the control gain and $u_e \in \mathbb{R}^2$ is a constant vector. A state and input coordinate transformation is applied by introducing $z(t), v(t)$, defined by

$$z(t) = x(t) - x_e,$$
$$v(t) = u(t) - u_e.$$

Consequently, the closed-loop form of the system (1) with the control strategy (4) becomes

$$z(t+1) = \max\{(a+bk)z(t) + w(t), -x_e\}. \tag{5}$$

Contrary to the nominal, disturbance-free case, for the actual system (5), each operating condition refers inevitably to a *set* of average response times x rather than a singleton due to the presence of additive disturbances. This set is known in the control literature as the *minimal robust positively invariant set* or the *0-reachable set* [3, 23]. It represents the set of states that can be reached from the equilibrium point under a bounded disturbance.

Definition 1. Consider system (5). An interval $\mathcal{S} = [s_m, s_M]$ is called an *invariant set*[1] for system (5) if $z(0) \in \mathcal{S}$ implies $z(t) \in \mathcal{S}$, for all $t \geq 0$ and any $w(t) \in [w_m, w_M]$. If, additionally, $|a + bk| < 1$, an interval $\mathcal{S}_{\min}$ is called the *minimal invariant set* with respect to (5) if it is invariant and it is included in any other invariant interval. Last, consider the constraints $z(t) \in \mathcal{Z} = [z_m, z_M]$. The interval $\mathcal{S}_{\max} \subseteq \mathcal{Z}$ is called the *maximal admissible invariant set* with respect to (5) if it is invariant and includes any other invariant interval.

Computing the minimal invariant set exactly is difficult, because in the general case it is the limit of a set sequence that converges only asymptotically. Nevertheless, in our case, since we opted to utilize scalar systems, it has an analytical description. This fact allows the simultaneous characterization of a stabilizing control gain and the minimal invariant set.

Theorem 1. *Let $x_e \in \mathbb{R}, u_e \in \mathbb{R}^2$ and $k \in \mathbb{R}^2$ satisfy (6) through (11):*

$$(1-a)x_e = bu_e, \tag{6}$$
$$0 \leq x_e \leq x_M, \tag{7}$$
$$u_m \leq u_e \leq u_M, \tag{8}$$
$$\frac{w_M}{1-a-bk} \leq x_M - x_e, \tag{9}$$
$$0 \leq a + bk < 1, \tag{10}$$
$$\max\left\{\frac{u_m - u_e}{x_M - x_e}, \frac{u_M - u_e}{-x_e}\right\} \leq k \leq \min\left\{\frac{u_M - u_e}{x_M - x_e}, \frac{u_m - u_e}{-x_e}\right\}. \tag{11}$$

The following hold.

(i) *The set*
$$\mathcal{S}_{\min} = \left[\max\left\{x_e + \frac{w_m}{1-a-bk}, 0\right\}, x_e + \frac{w_M}{1-a-bk}\right] \tag{12}$$
is the minimal robust positively invariant set with respect to the system (1) under state feedback (4).

(ii) *The set $\mathcal{S}_{\max} = \mathbb{X}$ is the maximal robustly invariant set with respect to the system (1) under state feedback (4).*

[1]By invariance, we mean robust positive invariance, or D-invariance (e.g., see [3, 15]).

(iii) *For any initial condition $x(0) \in \mathcal{S}_{\max}$ and any positive number ϵ, there is a time $T > 0$ such that*

$$\max_{y \in \mathcal{S}_{\min}} |x(T) - y| \leq \epsilon. \tag{13}$$

Proof.

(i) From (6) through (8), x_e is an admissible equilibrium point for the nominal system (1) with control input u_e. By (10) and Theorem 4.1 in Kolmanovsky and Gilbert [15], the minimal invariant set with respect to (5) is given by the limit of the forward reachable sets sequence. In our case, this sequence is defined by the iteration[2]

$$\mathcal{R}_0 = \{0\},$$
$$\mathcal{R}_{i+1} = ((a+bk)\mathcal{R}_i \oplus [w_m, w_M]) \cap [-x_e, \infty).$$

Since we are dealing with intervals, it is straightforward to see that for any $i \geq 0$,

$$\mathcal{R}_i = \left[\max\left\{\sum_{k=0}^{i-1}(a+bk)^i w_m, -x_e\right\}, \sum_{k=0}^{i-1}(a+bk)^i w_M\right],$$

and consequently, the minimal invariant set for the system (1) is directly given by (12).

(ii) Setting $z = x - x_e$, we show that $\mathcal{S}_{\max} = \mathcal{S}_1 \cap \mathcal{S}_2$, where $\mathcal{S}_1 := \{z : u_m - u_e \leq kz \leq u_M - u_e\}$ and $\mathcal{S}_2 := \{z : -x_e \leq z \leq x_M - x_e\}$. Specifically, we show that $\mathcal{S}_2$ is invariant and also $\mathcal{S}_2 \subseteq \mathcal{S}_1$. Since $\mathcal{S}_2$ is the translation of the state constraints $\mathbb{X}$ in z, the claim will be proved.

To this purpose, for $\mathcal{S}_2$, we first assume that $w = w_M$; then from (9), we get $w \leq (1-a-bk)(x_M - x_e)$. Considering the maximum value of $\mathcal{S}_2$ $z_0 = x_M - x_e$, then $z_1 = x_M - x_e$, $z_1 \in \mathcal{S}_2$. Accordingly, considering the minimum value of $\mathcal{S}_2$ $z_0 = -x_e$, then $z_1 = (1-a-bk)x_M - x_e$, and by applying (10), we still get that $z_1 \in \mathcal{S}_2$. Next, let us assume that $w = w_m$; as it stands, $w_m < w_M$ and so the aforementioned paradigm let us conclude that $z_1 \in \mathcal{S}_2$. Thus, by induction, we conclude that $-x_e \leq z_{t+1} \leq x_M - x_e$ while $z_t \in \mathcal{S}_{\max}$, for all $t \geq 0$ and any $w(t) \in [w_m, w_M]$.

To show $\mathcal{S}_1 \supseteq \mathcal{S}_2$, it suffices to show that $-x_e \in \mathcal{S}_1$ and $x_M - x_e \in \mathcal{S}_1$. Indeed, for $z_1 = x_M - x_e$, it holds that

$$\frac{u_m - u_e}{x_M - x_e} \leq k \leq \frac{u_M - u_e}{x_M - x_e},$$

whereas for $z_1 = -x_e$, we have that

$$\frac{u_M - u_e}{-x_e} \leq k \leq \frac{u_m - u_e}{-x_e}.$$

Both sets of the inequalities are satisfied due to the hypothesis (11). Consequently, $\mathcal{S}_1 \supseteq \mathcal{S}_2$, and since $\mathcal{S}_2$ is invariant, $\mathcal{S}_{\max}$ is invariant and admissible as well. Maximality of $\mathcal{S}_{\max}$ follows directly by observing that any $x_0 \notin \mathcal{S}_{\max}$ violates the state constraints (2).

(iii) We show that any trajectory beginning from $\mathcal{S}_{\max}$ is driven asymptotically (in fact, exponentially) to $\mathcal{S}_{\min}$. To this purpose, for any $z_0 \in \mathcal{S}_{\max}$ and then after i time intervals, it holds that

$$z_i = (a+bk)^i z_0 + \sum_{j=0}^{i-1}(a+bk)^j w_j, \tag{14}$$

where $w_j \in [w_m, w_M]$, $j = 0, \ldots, i-1$. By (10), the first term in (14) converges to zero exponentially, whereas the second term, as shown in (i), is bounded in $\mathcal{S}_{\min}$. Thus, given any

[2]For two sets $\mathcal{X}, \mathcal{Y}$, we have $\mathcal{X} \oplus \mathcal{Y} = \{x + y : x \in \mathcal{X}, y \in \mathcal{Y}\}$.

$\epsilon > 0$ and setting $a + bk = l < 1$, from (14) we have that since $z_0 \in \mathcal{S}_{\max}$, then necessarily $z_i \in l^i \mathcal{S}_{\max} \oplus \mathcal{S}_{\min}$. Consequently, $z_i \in \mathcal{S}_{\min}$.

Since $S_{\max}$ and $S_{\min}$ are intervals containing zero, we can always find a positive scalar d such that $S_{\max} = dS_{\min}$, and thus $z_{i+1} = (l^i d + 1)\mathcal{S}_{\min}$. Therefore, (13) can be satisfied for any T such that $(l^T d + 1)\mathcal{S}_{\min} \le (1 + \epsilon)\,\mathcal{S}_{\min}$, or, $T \ge \log_l \frac{\epsilon}{d}$. □

Remark 1. It is worth underlining that any choice of the control gain k that satisfies the relations (2) and (3) will render the whole constraint set as invariant.

Theorem 1 characterizes simultaneously the minimal invariant set and the gain of the associated control law. More importantly, it provides a tractable method of retrieving $S_{\min}$ and k. Specifically, for each model of (1) and given the equilibrium x_e, a feasible equilibrium pair (x_e, u_e), close to the pair of the desired values $(x_e, u_e^\star)$, and a state feedback control law of (4), which steers the closed loop system inside the minimal invariant set, can be calculated by solving the following linear programming problem,

$$\min_{u_e, k} \; \|u_e - u_e^\star\|_\infty \tag{15a}$$

subject to

$$(1 - a)x_e = bu_e \tag{15b}$$

$$u_m \le u_e \le u_M \tag{15c}$$

$$bk \le 1 - a - \frac{w_M}{x_M - x_e} \tag{15d}$$

$$-u_e - (x_M - x_e)k \le u_m \tag{15e}$$

$$u_e + x_e k \le u_M \tag{15f}$$

$$u_m \le u_e + (x_M - x_e)k \le u_M \tag{15g}$$

$$u_m \le u_e - x_e k \le u_M \tag{15h}$$

$$0 \le a + bk < 1, \tag{15i}$$

where constraint (15b) ensures that (x_e, u_e) is an equilibrium pair and (15c) means that the input constraints are satisfied. The constraint (15d), identically to (9), indicates that (x_e, u_e) belongs to $\mathcal{S}_{\min}$, whereas the constraints (15e) through (15h) are an analytical description of (11) ensuring that (x_e, u_e) belongs to $\mathcal{S}_{\max}$. Finally, the constraint (15i) is identical to (10).

Apart from the guarantee of the QoS metrics, the computed feasible operating points are used by the upper control level to determine the operating state of the activated VMs. The horizontal scaler, as it is described in Section 3.3.1, selects the operating area of each activated VM from the set of feasible operating points. Complementary to this, the local controller ensures that the chosen VM operating state will be realized by the described vertical scaling approach.

3.3 Horizontal Scaling

As discussed earlier, the upper control level consists of two essential components: the horizontal scaler and the workload predictor. The former aims to implement the appropriate VM placement on the minimum number of active edge servers, to satisfy the total workload of the co-hosted applications. The latter estimates the workload for the following time window, based on the previous actual value measured. This control level considers a cluster of edge servers located in a single place. Load balancing between geographically dispersed edge server clusters is not goal of this work but is part of our future work.

3.3.1 Horizontal Scaler. The horizontal scaler aims to compromise the mutually exclusive goals of performance and resource utilization. In particular, since the edge servers' resources are not abundant, unregulated performance demands for a single application would require the high allocation of computational resources on all servers, leaving the co-hosted applications in resource starvation. This is not desirable if the QoS requirements are met with less resources. The horizontal scaler component is responsible for optimizing the VMs' instantiation and for distributing the total requests of the implemented applications among them. The optimization objective of our approach is to minimize the number of the active edge servers, with the constraint of meeting the total workload demands. This indirectly results in reducing the consumed energy and optimally allocating the resources in the server side. The proposed horizontal scaler component leverages the fact that the size of a cluster of edge servers is small compared to a cloud datacenter, and thus a heuristic solution can be reached with small computation effort. In our approach, we make the assumption that each edge server hosts at most one VM per application. Taking this into consideration, the horizontal scaler's functionality breaks down in two steps. At the first offline step, it computes all the feasible VM placements within a single server, based on the set of the VMs' feasible operating points. These feasible placements are the ones where the total CPUs and memory required from the co-hosted VMs' operating points do not exceed a predefined threshold. Since we do not consider memory as a control variable, a static portion of memory r_e is assigned to every feasible operating point. For example, assume two applications App^x and App^y; a VM running App^x and instantiated at an operating point that requires 25% allocated CPU and 4GB of RAM, alongside a VM running App^y and instantiated at an operating point, which requires 55% allocated CPU and 8GB of RAM, is a feasible VM placement for a single edge server, as the total allocated CPU and memory do not exceed the threshold C_E, set at the 90% of the server's total CPU capacity and R_E set 32GB of RAM, respectively. More formally, the set of all feasible VM placements is defined as

$$\mathcal{P} := \left\{ p_i = \left(\left(u_{1e}^1, r_e^1\right), \ldots, \left(u_{1e}^N, r_e^N\right), \right),\ i = 1, \ldots, N : \sum_{i=1}^{N} u_{1e}^i \leq C_E, \sum_{i=1}^{N} r_e^i \leq R_E \right\}.$$

Then, assuming this set $\mathcal{P}$, this set's cardinality $|\mathcal{P}|$ and the total number of the edge servers E, it determines the number of servers to be activated E_A, by solving the following mixed integer linear program in an online fashion,

$$\min_{f_i, E_A} \{E_A\}, \tag{16a}$$

subject to

$$f_i \geq 0,\ i = 1, \ldots, |\mathcal{P}|, \tag{16b}$$

$$E_A = \sum_{i=1}^{|\mathcal{P}|} f_i, \tag{16c}$$

$$0 \leq E_A \leq E, \tag{16d}$$

$$\sum_{i=1}^{|\mathcal{P}|} f_i u_{2e}^j \geq \tilde{\Lambda}^j,\ j = 1, \ldots, N, \tag{16e}$$

where the positive integer variables f_i denotes how many servers with the p_i VM placement of set $\mathcal{P}$ need to be activated. As the constraint (16c) denotes, the sum of these variables is equal to E_A. The constraint (16d) simply restricts these activated servers to the total number of the edge servers. Finally, the last N constraints of (16e) denote that the estimated total workload for each application $\tilde{\Lambda}^j$, as it is computed in the following section, is satisfied by the selected VM placements. It is important to point out that the horizontal scaler component is triggered only if a significant variation in any of the application's workload occurs. This intends to avoid the frequent server activation/deactivation, which leads to oscillation of resource allocation and degradation of the VM's performance.

3.3.2 Workload Predictor. For each application, the total incoming RR is estimated by the Holt linear exponential smoothing filter [21] that captures the linear trend of time series. For any time interval i, the one-step prediction $\tilde{\Lambda}(i)$ of the incoming RR $\Lambda(i)$ is

$$\begin{aligned} \tilde{\Lambda}(i) &= \hat{\Lambda}(i) + c(i), \\ \hat{\Lambda}(i) &= \alpha\Lambda(i) + (1-\alpha)(\hat{\Lambda}(i-1) + c(i-1)), \\ c(i) &= \beta(\hat{\Lambda}(i) - \hat{\Lambda}(i-1)) + (1-\beta)c(i-1), \end{aligned} \tag{17}$$

where α, β are smoothing constants, $\hat{\Lambda}(i)$ is the smoothed value, and $c(i)$ denotes the linear trend in the measurement series. For the initialization, a random value of $\hat{\Lambda}(0)$ is used within the range of the incoming RR and $c(0) = 0.5$.

4 EVALUATION

In this section, we present an experimentation on the proposed computation offloading mechanism and the respective results. These results illustrate the success of our approach in guaranteeing the stability of application response times within an acceptable margin. We highlight the optimization of the resource allocation in terms of edge servers activated to serve the incoming workload. Moreover, an experimental comparison between the vertical scaling part of our mechanism and that of Xia et al. [30] is demonstrated. The benchmarking is performed using CloudSim Plus [27], a simulation environment suitable for CC and MCC experimentation, on a dual-core, macOS-powered system with 8GB of available memory.

4.1 The Horizontal Scaler's Complexity

Before proceeding with the detailed presentation of the experimental setup used throughout our detailed evaluation study and the presentation of the corresponding performance of the proposed computation offloading mechanism, we present some initial numerical results regarding the complexity of the horizontal scaler. As expected, the problem we are solving is a combinatorial one expressed as a mixed integer linear program in (16). For treating the mixed integer problem of the horizontal scaler, the GLPK solver [9] is used. The problem under consideration is generally NP-hard, and the lower bound of the computational complexity of the branch-and-cut algorithm used to find a solution is exponential [6]. Specifically, in the following, we analyze the performance of the horizontal scaler considering the dominant parameters of the optimization problem: the number of mobile applications, the total number of the feasible operating points of all applications, and the number of available edge servers.

Figure 3 illustrates the effect of the preceding parameters. The left graph demonstrates the effect of the number of the feasible operating points. Three applications are co-hosted in a cluster of servers, and the number of available operating points per application varies from 3 to 6, which produces a set $\mathcal{P}$ with a cardinality of 27 to 116, respectively. Subsequently, the computational time of (16a) increases accordingly. The middle graph of Figure 3 shows that the computational time also increases as the consolidated applications grow in numbers. More applications lead to more operating points, and consequently to the exponential increase of the computational time. Finally, at the right side of Figure 3, the effect of the number of the available edge servers is illustrated. As observed by the corresponding results, this parameter substantially affects the computational time only when the number of active edge servers is high. However, it should be noted that mobile

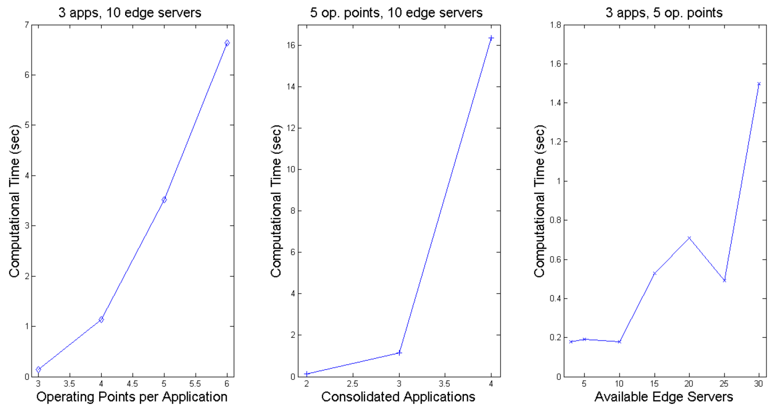


Fig. 3. Analysis of the horizontal scaler's computational complexity.

Table 2. VM Operating Points $(x_e, u_{1e}, u_{2e}, r_e)$

VMs of App^1	VMs of App^2
$(0, 0, 0, 0)$	$(0, 0, 0, 0)$
$(3, 25, 2.95, 4)$	$(3.75, 25, 3.23, 4)$
$(3, 35, 4.63, 6)$	$(3.75, 35, 5.29, 6)$
$(3, 45, 6.18, 6)$	$(3.75, 45, 7.38, 6)$
$(3, 55, 8.02, 8)$	$(3.75, 55, 9.58, 8)$

edge computing, contrary to the traditional cloud environment, is usually based on small/medium data centers that typically are expected to host few applications.

4.2 Experiment Setup

In our simulation, which spans around 4 hours and 10 minutes, or 15,000 seconds, we assume three physical machines with 32GB of RAM that are utilized as edge servers; as mentioned earlier, each of them is manually restricted to hosting at most two isolated VMs, each of which realizes one of the two supposed applications, $(N = 2)$, named App^1 and App^2; the edge servers are also restricted to hosting no more than one VM per application. More specifically, we follow this notation: VM_{ij} corresponds to the VM running on the i_{th} server and implementing the j_{th} application. The mobile traffic is simulated with a Poisson distribution of requests arriving at the horizontal scaler component, whereas the length of each request follows an exponential distribution. For both applications, the incoming offload RR varies between 1 and $25 req/sec$. However, for each of the application-specific VMs, the distributed RR range is divided in the following four subintervals: $[0, 3.5]$, $[3.5, 5.5]$, $[5.5, 7.5]$, and $[7.5, 10]$. A model of (1) is identified, and an equilibrium point and a control law are computed by solving (15a) through (15i) for every subinterval. Thus, in total, we identify offline eight systems and their respective controllers. The worst acceptable response time for the offloaded requests is set to 6 seconds and 7.5 seconds for App^1 and App^2, respectively. The desired average response time of the equilibrium points of the applications are set to the half of these values, $x_e^1 = 3$ and $x_e^2 = 3.75$. Indicatively, Table 2 depicts the operating points computed for both applications $(x_e^i, u_{1e}^i, u_{2e}^i, r_e^i)$. The first operating point, with zero input and average response time, corresponds to an inactive VM. Table 2 also justifies our assumption of hosting only one VM per application per server. For example, co-locating two VMs of App^1, namely running on the

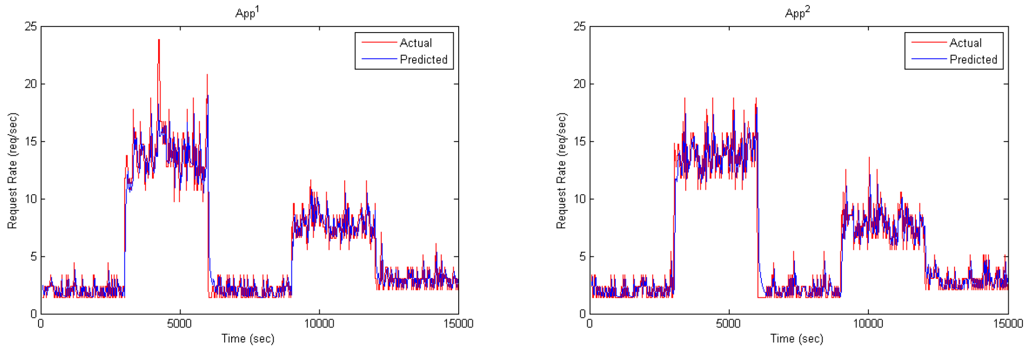


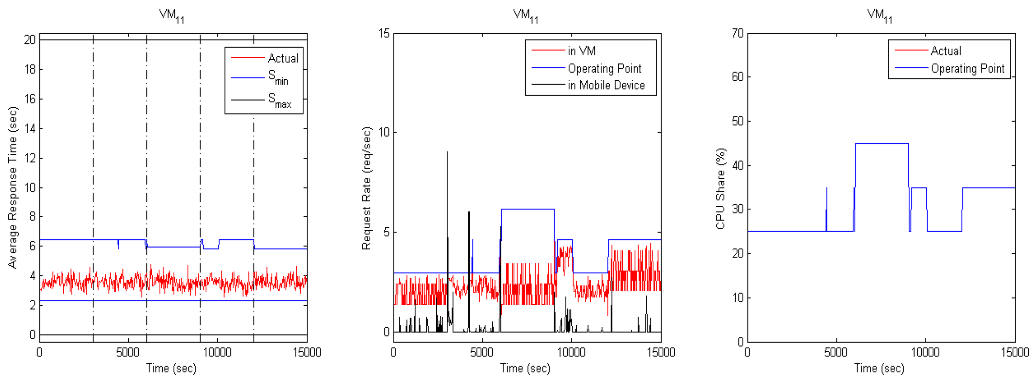
Fig. 4. Incoming offloading RRs for both App^1 and App^2.

second operating point of Table 2, would result in cumulatively serving less offloaded requests on average than deploying a single VM running on the fourth operating point, although the latter choice would result in allocating less CPU. This is a consequence of the related operating overhead of each separate VM deployment.

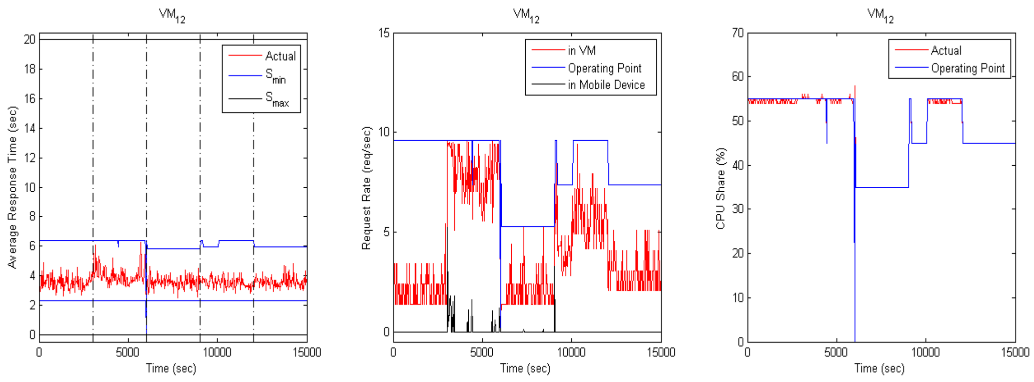
As described earlier, at the end of the time window (i.e., every 30 seconds), the workload predictor estimates the incoming RR for the next window. When the previously predicted RR and the currently predicted RR have an absolute difference greater than a predefined threshold, specific to the nature of this application, the horizontal scaler is triggered and selects the appropriate VM placement to be instantiated at the edge servers. For the particular applications, this threshold is set at 3req/sec. The duration of this time window is selected after considering the maximum time it would take for an in-range user to take the decision to offload, connect, offload, and receive the results. During this window, the RR remains relatively stable. A much larger time window would fail to adapt to the changes in the RRs, whereas a much shorter time window would probably result in unnecessary invocations. Furthermore, the control interval of 30 seconds appears to be adequate for the computation of the VM placement by the horizontal scaler, as it is later shown in Section 4.3. However, for other types of applications, this control time interval may be selected differently: on one hand, to be larger than the computational time of mixed integer problem that needs to be solved, and on the other hand, efficient enough to properly follow the variation of the incoming requests.

4.3 Numerical Results

The results depicted in Figures 4 through 8 are used to evaluate the efficiency of our proposed mechanism in the preceding scenario. Figure 4 depicts the fluctuations in the actual (red line) and the predicted (blue line) RR per application during the experiment. For both applications, the actual incoming RR is altered every 50 minutes, or 3,000 seconds. Figures 5 through 7 illustrate the measured average response time and the inputs per VM in each server, respectively; the left graph of each figure depicts the actual application response time (red line) together with the boundaries of the positive invariant sets, $\mathcal{S}_{min}$ (blue line) and $\mathcal{S}_{max}$ (black line). The middle shows the actual RR served by the respective VM on the edge server (red line), together with the RR rejected by the particular VM and sent back to the mobile device for execution (black line). The nominal RR value of the selected VM's operating point is also shown (blue line). In the right graph, the actual CPU share allocated to the VM is shown (red line), alongside the operating point's nominal value (blue line), for each given moment.



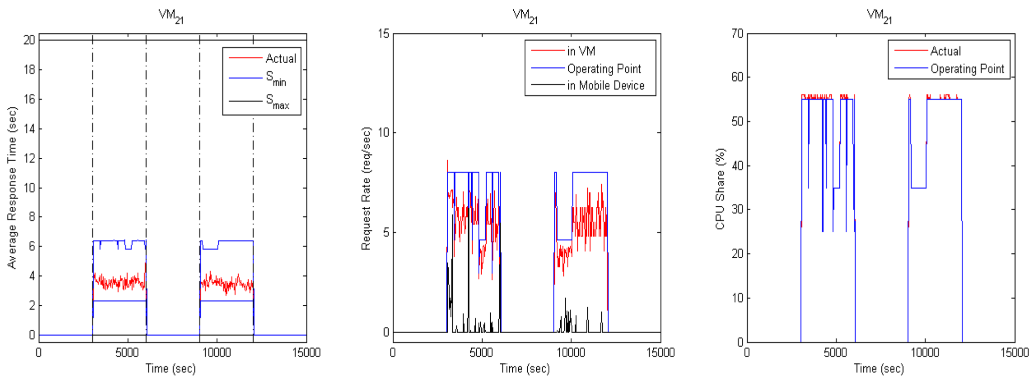
(a) VM_{11} Performance.



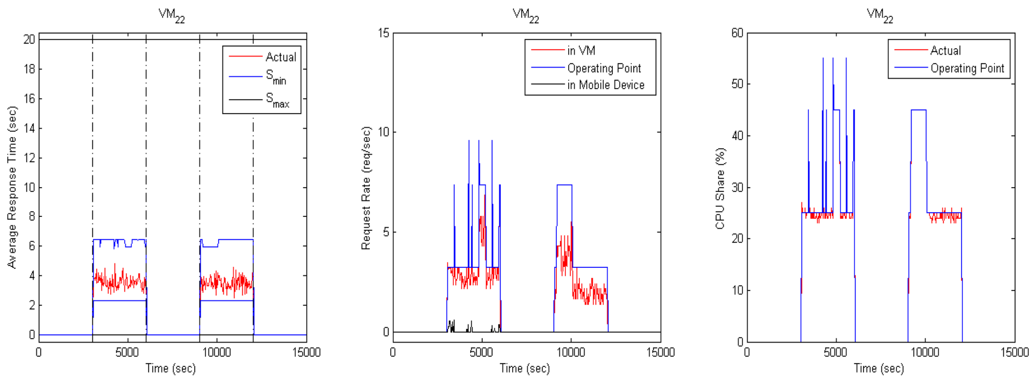
(b) VM_{12} Performance.

Fig. 5. Average response time, RR, and CPU share allocation of VMs in the first server.

We can observe, in the left graph of Figures 5(a), 6(a), and 7(a), that the average application response time for App^1 remains between the given constraints, despite the workload fluctuation. The similar results are observed in the left graph of Ffigures 5(b), 6(b), and 7(b) for App^2. This means that the theoretical guarantees of Theorem 1 (i) are translated in the response times not exceeding the boundaries of the minimal invariant set, $\mathcal{S}_{\min}$ and $\mathcal{S}_{\max}$. The middle graphs of Figures 5 through 7 depict how the horizontal scaler adapts to these fluctuations and selects the appropriate placement, in terms of number of active edge servers, VMs, and their operating points, to meet the demanded RR. As shown in Figure 8, it activates one edge server between 0 and 3,000 seconds, 6,000 and 9,000 seconds, and 12,000 and 15,000 second, and two edge servers between 9,000 and 12,000 seconds and between 3,000 and 6,000 seconds. Of these incoming workload fluctuations, the rapid ones (e.g., around the 3,000-second area) allow us to also demonstrate the local controllers' functionality; in such situations, the workload predictor component requires a time window to adapt because the estimated RR value is based on the previous actual incoming RR value. This results in the horizontal scaler failing to select the appropriate VM placement for the specific time window. However, each VM's local controller proves to be en garde by rejecting the excessive offloading requests and redirecting them back to the mobile device for execution, to guarantee the stability of the response time. This guarantee is also provided by the local controller in the form of vertically scaling the VM; the workload predictor's minor inaccuracies are handled by moderately regulating



(a) VM_{21} Performance.

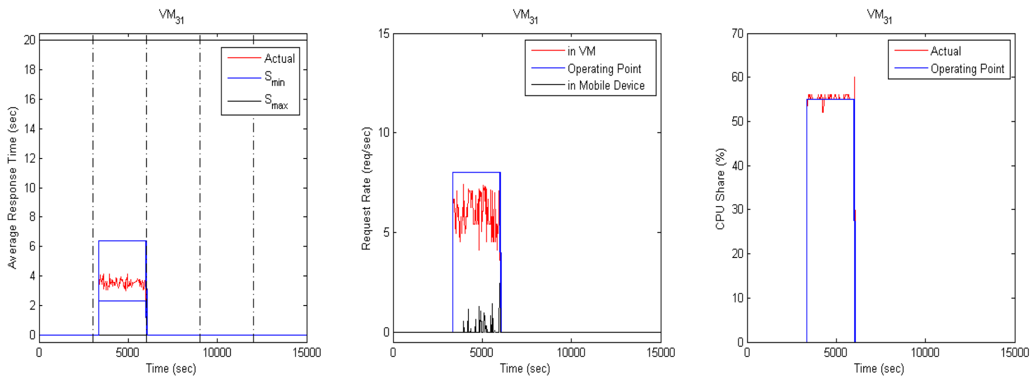


(b) VM_{22} Performance.

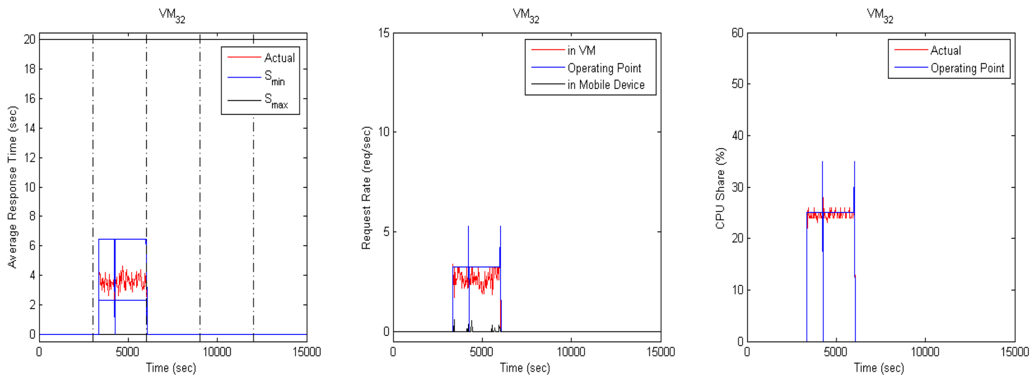
Fig. 6. Average response time, RR, and CPU share allocation of VMs in the second server.

the CPU resources and the accepted RR within limits of the operating point's area. This procedure is illustrated in the middle graph of each subfigure of Figures 5 through 7; when the RR accepted in the VM has reached the value calculated from the local controller for the selected operating point, the excessive, rejected RR, which as a consequence is relocated to the mobile devices for execution, is increased. Also at the third graph of each subfigure, where some minor fluctuations are observed, in the actual CPU share from the respective nominal values of the operating point. It is important to remark that for every VM and for the most part of the experiment, the actual and the nominal values of the CPU share overlap, making only the blue line observable. Furthermore, some short sudden changes in the selected operating points of the VMs, depicted in the second and third graphs of the subfigures, occur due to certain spikes in the incoming RR; these spikes are so acute that the horizontal scaler's trigger condition is satisfied. Consequently, the appropriate VM placement is recalculated with the updated operating points. We can see that it is this combination of horizontal and vertical scaling that results in the overwhelming majority of offloading requests being successfully served; 95.18% of the total requests for App^1 and 98.74% for App^2, respectively.

Another interesting remark is that the horizontal scaler selects a VM placement, which minimizes the number of active servers but not necessarily the total allocated CPU share. This happens due to the structure of the optimization problem's objective function in (16a). One approach to additionally include this optimization objective in our framework would be to revert to



(a) VM_{31} Performance.



(b) VM_{32} Performance.

Fig. 7. Average response time, RR, and CPU share allocation of VMs in the third server.

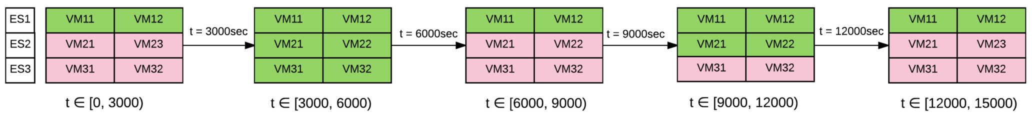


Fig. 8. Active VMs in edge servers.

multiobjective optimization, either by using preemptive optimization or a multiobjective cost. However, this would significantly increase the time complexity of the decision-making part without envisioning substantial benefits.

4.4 Comparative Results

A second experiment better demonstrates the performance of the proposed vertical scaling mechanism alone and compares it to that in Xia et al. [30]. This is an energy-aware offloading approach, which uses edge server VMs with fixed CPU shares allocated. The offload decision depends on an SLA threshold for the response time of the offloaded requests, named T_d. At the end of each time window (i.e., every 30 seconds), an estimation of the incoming RR for each application is computed by (17) and the input vector is updated according to (4), regarding the following window. The upper left graph of Figure 9 depicts the actual response time and the boundaries of $\mathcal{S}_{\min}$ and

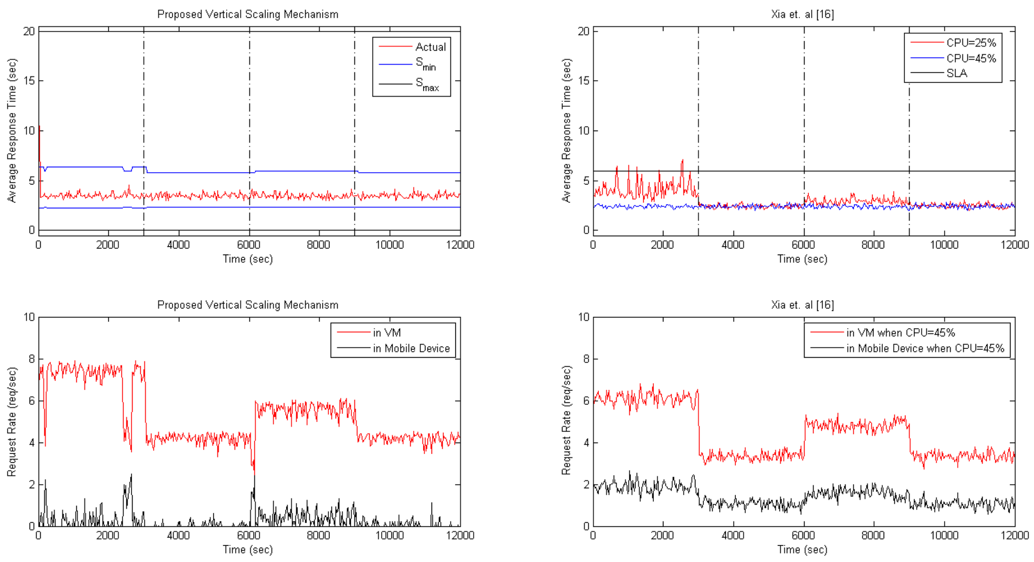


Fig. 9. Evaluation and comparison of proposed approach with Xia et al. [30].

$\mathcal{S}_{\max}$ for $App1$. After the initial interval, the response time steers from $\mathcal{S}_{\max}$, which we remind is equal to $\mathbb{X}$, to $\mathcal{S}_{\min}$ and remains within. This proves the validity of Theorem 1 (iii). In particular, by computing the control law solving the linear program (15a) through (15i), we see the convergence to the minimal invariant set. The upper right graph shows the average response time for allocated $CPU = 25\%,\ 45\%$, and $T_d = 6$ of the approach [30]. In the first quarter of this graph, the SLA is violated for the underprovisioned VM with $CPU = 25\%$. The second row of Figure 9 again illustrates the RRs served by the edge server and the mobile devices. On the left side, our approach seems to adapt well against the various incoming RR. Once again, the observed rapid fluctuation of the served requests exists due to false predictions of the incoming RR. As expected, this does not affect the response time. On the right side, it seems that the use of T_d restricts the amount of requests directed to the edge server. This explains the better response times of the upper right graph. For our proposed offloading mechanism, the requests served at the edge server approach 95.54% of the whole workload, whereas for Xia et al. [30], this percentage is limited to approximately 76%, for both CPU shares. It is clear that our proposed approach performs better against the varying workload because of the vertical scaling of the VMs.

5 CONCLUSION AND FUTURE WORK

In this study, a cooperative, two-level computation offloading mechanism for mobile applications is presented. The VM operation is modeled by a group of LTI models, and for each model, an equilibrium operating point, a proper controller, and the minimal and maximal positive invariant sets are computed. At the upper level, a horizontal scaling procedure takes place; an optimizer determines the number of active edge servers and the operating points of the VMs to be implemented in them, to serve the total workload for each application. This decision takes into consideration the calculated equilibrium points for each underlying VM, thus guaranteeing the scalability of our mechanism toward major workload fluctuations. At the local level, a controller handles the minor workload fluctuations by scaling the VMs vertically, ensuring that the average response time is stabilized and restricted in a specific range of values. The experimental evaluation shows that the proposed mechanism achieves a high percentage of requests admitted in the edge servers while the performance constraints are met, outperforming a well-established energy-aware offloading method.

Future work will focus on further investigating improvements on the modeling and control of the application-specific VMs and leveraging different combinatorial optimization criteria to improve the horizontal scaler's decision-making mechanisms. Specifically, it should be noted that, as mentioned before, in this work, we mainly aim at minimizing the number of active servers with the constraint of meeting the total workload demands. By offloading as many tasks as possible while keeping the number of active servers low, implicitly energy efficiency on the mobile nodes and the edge servers is targeted as well. However, dealing with explicitly optimizing energy cost in the mobile nodes (e.g., maximize the offloaded requests) or the edge servers (e.g., minimize number of allocated CPUs) is also an interesting and challenging problem and part of our current and future work. Additionally, minimizing functional costs like data transmission costs (e.g., how the requests are distributed among the servers) or maximizing revenue/income for the infrastructure providers (e.g., how many different VM-applications can run per server) can be used as additional or alternative objectives for the horizontal scaler component of our framework.

Furthermore, as mentioned in the beginning, user mobility has not been considered in this work. Nevertheless, this is a very challenging and important point, and we currently investigate the consideration and impact of user movement within an area covered by several wireless access points connected to an edge server cluster. In the same direction, the potential use of the proposed architecture and placement in the context of multiple proximate edge clouds, to accommodate workload balancing between edge servers located in remote areas, or between edge servers and the cloud, is an issue of high practical and research importance. Finally, we intend to test the proposed approach under real-life use cases, such as IoT- or 5G-enabled applications, in heterogeneous future Internet testbeds/infrastructures.

REFERENCES

[1] Marco V. Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. 2013. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In *Proceedings of the 2013 IEEE INFOCOM*. IEEE, Los Alamitos, CA, 1285–1293.

[2] Arani Bhattacharya and Pradipta De. 2017. A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications* 78 (2017), 97–115.

[3] Franco Blanchini and Stefano Miani. 2008. *Set-Theoretic Methods in Control*. Springer.

[4] Zhen Cao and Panagiotis Papadimitriou. 2016. Collaborative content caching in wireless edge with SDN. In *Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks*. ACM, New York, NY, 6.

[5] Valeria Cardellini, Vittoria De Nitto Personé, Valerio Di Valerio, Francisco Facchinei, Vincenzo Grassi, Francesco Lo Presti, and Veronica Piccialli. 2016. A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming* 157, 2 (2016), 421–449.

[6] Sanjeeb Dash. 2005. Exponential lower bounds on the lengths of some classes of branch-and-cut proofs. *Mathematics of Operations Research* 30, 3 (2005), 678–700.

[7] Dimitrios Dechouniotis, Nikolaos Leontiou, Nikolaos Athanasopoulos, George Bitsoris, and Spyros Denazis. 2012. ACRA: A unified admission control and resource allocation framework for virtualized environments. In *Proceedings of the 8th International Conference on Network and Service Management*. IEEE, Los Alamitos, CA, 145–149.

[8] Dimitrios Dechouniotis, Nikolaos Leontiou, Nikolaos Athanasopoulos, Athanasios Christakidis, and Spyros Denazis. 2015. A control-theoretic approach towards joint admission control and resource allocation of cloud computing services. *International Journal of Network Management* 25, 3 (2015), 159–180.

[9] GLPK-YALMIP. 2016. Mixed-Integer Linear Programming Solver. Retrieved June 24, 2018 from https://yalmip.github.io/solver/glpk/.

[10] Dinh Thai Hoang, Dusit Niyato, and Ping Wang. 2012. Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In *Proceedings of the 2012 IEEE Wireless Communications and Networking Conference (WCNC'12)*. IEEE, Los Alamitos, CA, 3145–3149.

[11] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S. Tucker. 2016. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications* 34, 5 (2016), 1728–1739.

[12] Mike Jia, Weifa Liang, Zichuan Xu, and Meitian Huang. 2016. Cloudlet load balancing in wireless metropolitan area networks. In *Proceedings of IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications*. IEEE, Los Alamitos, CA, 1–9.

[13] Haleh Khojasteh, Jelena Misic, and Vojislav Misic. 2016. Prioritization of overflow tasks to improve performance of mobile cloud. *IEEE Transactions on Cloud Computing* 7, 1, 287–297.

[14] Abbas Kiani and Nirwan Ansari. 2017. Optimal code partitioning over time and hierarchical cloudlets. *IEEE Communications Letters* 22, 1, 181–184.

[15] Ilya Kolmanovsky and Elmer G. Gilbert. 1998. Theory and computation of disturbance invariant sets for discrete-time linear systems. *Mathematical Problems in Engineering* 4, 4 (1998), 317–367.

[16] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18, 1 (2013), 129–140.

[17] Nikolaos Leontiou, Dimitrios Dechouniotis, Nikolaos Athanasopoulos, and Spyros Denazis. 2014. On load balancing and resource allocation in cloud services. In *Proceedings of the 2014 22nd Mediterranean Conference on Control and Automation (MED'14)*. IEEE, Los Alamitos, CA, 773–778.

[18] Nikolaos Leontiou, Dimitrios Dechouniotis, Spyros Denazis, and Symeon Papavassiliou. 2018. A hierarchical control framework of load balancing and resource allocation of cloud computing services. *Computers and Electrical Engineering* 67 (2018), 235–251.

[19] Mengyu Liu and Yuan Liu. 2017. Price-based distributed offloading for mobile-edge computing with computation capacity constraints. arXiv:1712.00599.

[20] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. 2018. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*. Springer, 103–130.

[21] Spyros Makridakis, Steven C. Wheelwright, and Rob J. Hyndman. 2008. *Forecasting Methods and Applications*. John Wiley & Sons.

[22] Hassan Raei and Nasser Yazdani. 2017. Analytical performance models for resource allocation schemes of cloudlet in mobile cloud computing. *Journal of Supercomputing* 73, 3 (2017), 1274–1305.

[23] Sasa V. Rakovic, Eric C. Kerrigan, Konstantinos I. Kouramas, and David Q. Mayne. 2005. Invariant approximations of the minimal robust positively invariant set. *IEEE Transactions on Automatic Control* 50, 3 (2005), 406–410.

[24] Zohreh Sanaei, Saeid Abolfazli, Abdullah Gani, and Rajkumar Buyya. 2014. Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *IEEE Communications Surveys and Tutorials* 16, 1 (2014), 369–392.

[25] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing* 8, 4 (2009), 14–23.

[26] Mahadev Satyanarayanan, Rolf Schuster, Maria Ebling, Gerhard Fettweis, Hannu Flinck, Kaustubh Joshi, and Krishan Sabnani. 2015. An open ecosystem for mobile-cloud convergence. *IEEE Communications Magazine* 53, 3 (2015), 63–70.

[27] Manoel C. Silva Filho, Raysa L. Oliveira, Claudio C. Monteiro, Pedro R. M. Inácio, and Mário M. Freire. 2017. CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM'17)*. IEEE, Los Alamitos, CA, 400–406.

[28] Xiang Sun and Nirwan Ansari. 2017. Avaptive avatar handoff in the cloudlet network. *IEEE Transactions on Cloud Computing* (2017). To be published.

[29] P. E. Wellstead and M. B. Zarrop. 1991. *Self-Tuning Systems: Control and Signal Processing*. John Wiley & Sons.

[30] Qiufen Xia, Weifa Liang, Zichuan Xu, and Bingbing Zhou. 2014. Online algorithms for location-aware task offloading in two-tiered mobile cloud environments. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC'14)*. IEEE, Los Alamitos, CA, 109–116.

[31] Zichuan Xu, Weifa Liang, Wenzheng Xu, Mike Jia, and Song Guo. 2015. Capacitated cloudlet placements in wireless metropolitan area networks. In *Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN'15)*. IEEE, Los Alamitos, CA, 570–578.

[32] Yang Zhang, Dusit Niyato, and Ping Wang. 2015. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing* 14, 12 (2015), 2516–2529.

Received December 2017; revised August 2018; accepted October 2018