SLA-driven VM Scheduling in Mobile Edge Computing

Kostas Katsalis*, Thanasis G. Papaioannou†, Navid Nikaein*, Leandros Tassiulas‡

* Institut Eurecom, Mobile Communications Dept., Sophia Antipolis, France
† Centre for Research and Technology-Hellas, CERTH-ITI, Greece
‡Yale University, Dept. Of Electrical and Computer Engineering,USA
*{kostas.katsalis, navid.nikaein}@eurecom.fr, †thanasis.papaioannou@iti.gr, ‡leandros.tassiulas@yale.edu

***Abstract*—Mobile-Edge Computing (MEC) is about offering application developers and service providers cloud-computing capabilities and an IT service environment at the edge of the mobile network. However, although cloud computing can be used to meet traditional challenges, like scalability concerns and provide for fast resource provisioning times, a multifaceted analysis is required when it comes in multi-operator environments with time-critical applications and services. In this work, we claim that the service importance must be at the epicenter when it comes to the scheduling and placement decision of whether to deploy the service at the edge network or not. Virtual machine (VM) scheduling decisions should avoid SLA violations for popular or time-critical services, and be fair between the service providers. A Lyapunov optimization framework is derived to solve this stochastic optimization problem that aims to maximize the revenue of the physical infrastructure owner in a multi-network operator-sharing environment with time-critical SLAs. A series of simulation experiments validate the high effectiveness of the proposed approach over benchmarking ones.**

***Keywords*-VM scheduling, Mobile Edge Computing, Stochastic Optimization, Service Differentiation, Mobile Cloud**

I. Introduction

The ever-increasing popularity and requirements of mobile applications, augment the need for efficient mobile computing architectures and efficient utilization of the wireless network physical resources. The core idea in the MEC design paradigm [1], [2] is to improve the end-user experience by utilizing cloud computing technologies and virtualized IT-based resources at the edge of the network. These are used in order to facilitate rapid service and function deployment, and the delivery of Over-The-Top (OTT) applications closer to the user. The concept considers multiple Radio Access Network (RAN) nodes (e.g., eNodeB, Wi-Fi etc.) with aggregated computing/memory/storage power in local nano-data centers, hierarchically located above the RAN (see Fig. 1). The idea is that a request for service or function must be handled at the edge cloud; the request is forwarded to some external cloud by the gateway systems only if necessary. Note that the interfaces of a detailed MEC architecture are not yet standardized [3].

Given the resource constraints in the edge network and the highly-volatile service demand, providing differentiated performance Service Level Agreements (SLAs) to various tenants and service providers that compete for the edge network resources is not trivial. There is some previous work on SLA-driven VM placement in the generalized cloud (like [4], [5], [6]), mostly based on demand forecasting for resource scheduling. However, in contrast to general cloud computing environments, in MEC, it is the limitation on the number of physical server resources that requires for extreme efficiency and proper modeling. Having this in mind, while also considering that a) multiple operators and stakeholders come into play concurrently to massively offer services, and b) there is lack of distinction between time-critical and non-time-critical services, the area of SLA-driven VM scheduling in the MEC context remains highly unexplored.

In this paper, we study a VM scheduling and placement problem, where the VMs are used to support services deployment at the edge network. In our approach, it is the SLA requirements set on a per provider basis, that drives the service deployment and the relevant VM instrumentation. Intuitively, a service with strict performance requirements must be preferred for deployment at the edge network, as compared to a service with loose performance requirements. In the case of multiple providers with a mixture of time-critical and non-time-critical services, a non-trivial scheduling problem arises for the MEC-IaaS provider, i.e., which service(s) from which provider(s) and where to be deployed at the edge. In our devised approach, the goal of the decision process is to maximize the MEC-IaaS provider revenue, while maximizing QoE for the customers of the time-critical services.

In more detail, the MEC-IaaS provider orchestrates rapid installation of services deployed on VMs for various service providers or mobile operators. A service provider can be any stakeholder that a client is associated with, in order to receive services (e.g., its mobile telecom operator). Henceforth, we will use the term mobile operators interchangeably with the service providers. A connected customer may request for services that are deployed in VMs that reside at the edge network or at some external cloud. In the former case, there is no need to create egress traffic beyond the edge network for receiving the services; hence, the delay is minimum and the QoE can reach to its extreme. Note that, potentially, the service to be deployed can be a web service or application or even a Virtual Network Function (VNF) (e.g., S-GW of the EPC network in LTE). This type of generality gives us the flexibility to apply the scheduling principles in various RAN/network

resources sharing concepts. A VM of a specific type at the edge cloud is assumed to be able to handle efficiently a certain workload for a specific service. Excess workload is routed to a VM of the service at an external cloud and results to a SLA violation, which is penalized with a certain fee.

We model the service/VM requests made by the mobile operators as a queuing system. Our scheduling approach selects for every mobile operator a number of VMs of specific types to place at the edge network taking into account a) the SLAs of the services that need to be scheduled, b) the physical limitations of the edge-cloud nodes and c) fairness among service providers. The number of requested VMs per service provider is dynamically adapted to the workload of the services of the provider over time. At each slot, if there are no available resources at the edge cloud, the excess VMs requested are deployed to some external cloud system.

The main contributions of this paper are as follows:

- We formulate a joint optimization problem where the MEC-IaaS provider aims to maximize its expected lifetime revenue when deploying VMs that host services and minimize SLA violations for the deployed services, while being fair among the service providers.
- The problem formulation takes into account not only the service requests by the service providers, but also the expected workload in the short run. The idea is to deploy VMs that host popular services.
- Due to stochastic nature of the problem, we employ Lyapunov optimization [7], [8], [9], in order to derive fast myopic VM placement that is close to optimal in the long run.
- Because of the power of Lyapunov optimization, we make no assumptions on the arrival rate or the service lifetime distributions, which are considered unknown. Thus, our methodology can be used to derive optimal bounds for any arbitrary arrival and lifetime distributions, as long as the relevant random variables that constitute the stochastic nature of the problem are bounded.
- We evaluate our approach by extensive simulations of the proposed solution against variations of well-known benchmark techniques (i.e., First Fit).

We believe that our work can be used in multiple application scenarios and various 5G use cases, like the ones identified by NGMN [2]. Also, note that, although we target the MEC environment, our scheduling and placement approach is quite generic and it can be employed in different cloud settings.

The rest of the paper is organized as follows: in Section II, we present the system model, whereas in Section III, we formally state the problem under consideration. In IV, we present the proposed optimization approach for SLA-driven VM scheduling. In Section V, we evaluate our approach by means of simulation experiments. In Section VI, we overview the related work. We conclude our paper in Section VII.

II. System model

We assume a system with a set of $\mathcal{N} = \{1, 2, \ldots, N\}$ of servers at the edge network. Let $\mathcal{P} = \{1, 2, \ldots, P\}$ denote the

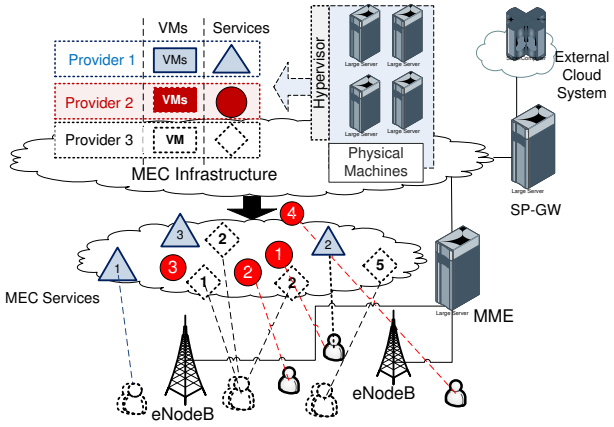


Fig. 1: The MEC concept in LTE networks and services offering by multiple different tenants (service providers).

set of Mobile Service Operators (e.g., MSO A, MSO B, etc.). Every wireless client in the area can be associated with such a provider, in order to receive services.

Controller operation: Our system operates in time slots. During each time slot, every provider sends requests to the cloud infrastructure owner (MEC-IaaS provider) to deploy a specific service. This service must be deployed over a VM at some server. The lifetime of deployment is also part of the request, as opposed to the VM type. Every request for some service with a certain SLA for performance is actually mapped to a VM request of a certain type by the controller (e.g., an advertisement service can be deployed in a "small" VM, while a time-critical service may need more powerful VMs). We explain the mapping process in detail in the following. For ease of notation, we assume a predefined set of available VM types (i.e., "small", "medium", "large") and let $\mathcal{V}$ denote the set with all the VM types. Also, let $\mathcal{R}$ be the set of capabilities of any physical or virtual machine, i.e., $\mathcal{R} = \{cpu, memory, storage, bandwidth\}$.

The control actions are taken at the beginning of each time-slot. At each control instant, the controller of the MEC-IaaS provider is responsible to decide: a) which new services are to be deployed, b) which VM type the service request is mapped to and c) in which server to deploy the VM at. If there is no available capacity, VMs are deployed at some remote cloud system. In our model, at the beginning of each time slot, the controller flushes all the enqueued requests for all the providers queues and no service backlog is transferred to the next slot. For simplicity and clarity of presentation, we refrain from considering VM migrations in this work. We plan to enhance our model and take into the account the possibility of VM migrations in the future.

The controller operation is based on the incentive to maximize the revenue of the MEC-IaaS provider by deploying the most popular (by means of users requests) and performance-demanding services, while utilizing edge-cloud resources as much as possible. The deployment of more performance-demanding services (or applications or even virtual functions) at the edge servers increases the QoE of end users and the

network utilization efficiency at the same time. This is a good motivation for the service provider to pay the MEC-IaaS provider to host its services. We explain this process in detail below.

Description of the Service Request process: From the business perspective, every provider requires the satisfaction of a specific SLA for each service. With our approach even very strict SLAs can be met by deploying the service at VMs at the actual edge, following an optimal strategy. Note that each VM instance that runs a specific service for a provider can serve up to a maximum number of client requests, while respecting the service SLA. Client service requests that cannot be handled by VMs at the edge incur a certain penalty payable to the service provider (e.g., similar to the penalty fee described in the Amazon EC2 SLA for service unavailability).

Let $\mathcal{S}_i = (S_{i,j})$ denote the set of all the services j the provider $i \in \mathcal{P}$ wants to deploy. Let $\mathcal{S}$ denote the set of all services $\mathcal{S} = \cup S_i$. Every service can be deployed in some VMs of type $v \in \mathcal{V}$. The service model can be quite extensive, meaning that the services deployed may be related to interactive services, batch requests, or even to VNFs. In this work, we do not get into this level of detail. We only consider the request pattern associated with each service.

Requests for Specific Service Instances: For every service $s \in \mathcal{S}$, there is a) the lifetime period of deployment and b) an associated request rate from mobile end users and/or other applications that are using it. The client request rate of a service can be either estimated by the IaaS controller based on prior measurements of client requests for this service deployed at the edge or some exterior cloud from mobile users and/or other applications, or set by the Mobile Service Operator (MSO) based on private estimates (e.g., MSO 1 makes an initial estimate of 1000k requests per sec for some service he wants to deploy). From a practical perspective, by using packet steering techniques or SDN methodologies, we can obtain per flow/service statistics. In either case, in the next slot, just before the control decision, the estimate of the client request rate per service is adjusted according to the actual load using various moving average techniques, such as Simple or Exponential Moving Average.

We denote as $r_j^s(t)$ the aggregated rate for the expected number of client requests made by all mobile users or applications/functions associated with service provider j for service s. Moreover, we denote as $\tilde{r}_j^s(t)$ the actual request rate by all mobile users or applications associated with service provider j for service s when s is hosted at the edge network and as $\hat{r}_j^s(t)$ when s is hosted in some remote cloud. Then,

$$r_j^s(t) = \tilde{r}_j^s(t) + \hat{r}_j^s(t)\,. \tag{1}$$

All these rates are calculated based on prior statistics at the end of each slot and are used by the IaaS controller for the control decision in the beginning of the next slot.

Mapping Service Requests to VM Requests: In order to satisfy the client request rate for some service, a number of VMs is required. This relation is provided by some function

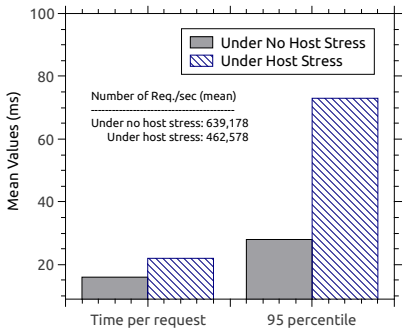


Fig. 2: Stressing Apache service at a host machine.

$f(\cdot)$, which is defined as follows:

$$f : \mathcal{RQ} \times \mathcal{S} \times \mathcal{P} \longrightarrow \mathbb{N}^{|\mathcal{V}|} \tag{2}$$

$f(\cdot)$ gives the vector of the numbers of VMs per VM type for handling requests $r_j^s(t) \in \mathcal{RQ}$ for service s according to the SLA requirements of customer j. For example, assume that provider j wants to deploy service s and it is expecting a request rate of $r_j^s(t) = 10000$ requests per slot, with some specific SLA requirements (e.g., 10ms average respond time). If function f returns the vector $(2, 1, 0)$, this means that this SLA can be satisfied with 2 small VMs or with 1 one medium VM. In this context, we also define as $\xi_j^{v,s}$ to be the maximum request rate for service s that can be served by a VM of type v within the delay bound per request satisfied by the SLA to customer j for service s. For example, as depicted in Fig. 2, we benchmark an Apache web server deployed on a VM, where we bring the CPU of the host machine in overload, using the Linux *stress* tool. In practice, this CPU overload of the host machine can be due to multi-VM operations or other functionality. As we can observe in Fig. 2, the maximum rate of requests that can satisfied under host CPU overload, within a 95-percentile delay bound of 73ms per HTTP request, is ξ=462.578 requests per second. That is, a decrease of almost 180 requests per sec from the case where the host is not stressed. For lower delay bounds or VMs with limited resources, ξ values for this service are expected to be lower. Employing similar stress tests per service s and per VM type v, one can find $\xi_j^{v,s}$ values for the delay bound per service s specified at the SLA of service provider j.

The problem of finally mapping service requests to VM requests resembles *Knapsack* as follows: Each VM of type v has a rent of w_v and serves up to $\xi_{v,s}$ client requests per slot. We need to serve W_s requests for service s. We employ a greedy approximation algorithm, which is a variation of [10], as follows: Sort the VM types in decreasing order of requests per unit of rent, $\xi_{v,s}/w_v$. Add the VM types to the sack, starting with as many copies as possible from the first VM type, so that adding one more VM of that VM type keeps the total number of requests satisfied lower than or equal to W_s. Then, continue adding VMs of the second VM type in the same manner and so on, until all VM types are examined. Finally, add one VM of the cheapest type to the sack, if the total number of satisfied requests is lower than W_s. Note that this last VM is guaranteed to increase the total number of

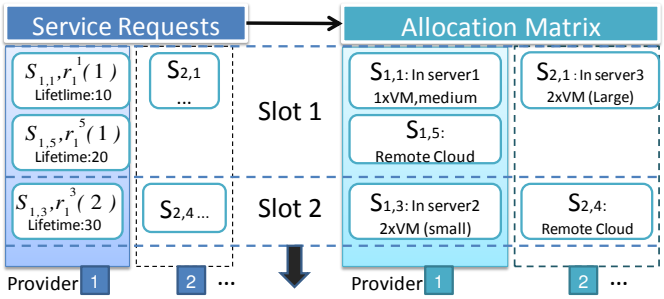


Fig. 3: An example allocation matrix.

satisfied requests to a value more than W_s. If the lowest total rent is m, then this greedy algorithm is guaranteed to achieve at least a value of $2 \cdot m$, while satisfying W_s client requests.

Capacity Constraints: A VM can be scheduled to some physical machine if its resource specification does not violate the physical capacity of the server in any of its resource pools. Let the vector $P_i = [p_i^j], j = 1, .., R$ denote the physical capabilities R. Any VM type $v \in \mathcal{V}$ defines a vector $M_v = [m_v^j], j = 1, .., R$ with the physical capabilities R required. Moreover, we define $n_{i,j}^{v,s}(t)$ as the number of VMs of type v that are hosted in physical machine i at time slot t for provider j and service s at the edge. Let $\boldsymbol{n}(t) = (n_{i,j}^{v,s}(t))$ be a vector with all the allocations of VMs in the edge cloud at time slot t. This is a state vector for our system.

See Fig. 3, for the outcome of the control scheduling and placement process. This is a cloud Allocation Matrix with the information of where to actually deploy and instantiate services on per provider and service basis.

Justification of the System Model: With the above analysis, we stress the fact that some services are more important than others and have different impact on the revenue that the service provider enjoys. By more important, we mean that more client requests are heading for these services or the service has time-critical requirements reflected in higher penalties for SLA violations. At the end of the time slot, if some service is not deployed at the edge, the MSO has the ability to just resend the request at the next time slot. Meanwhile, there is no service idle-time, since the service would be deployed in some external cloud.

III. Problem Statement

We define as $\boldsymbol{\omega}(t) = (\mathbf{A}(t), \mathbf{D}(t), \mathbf{r}(t))$ a random event at time slot t. A random event comprises the service/VMs request arrivals, the VM departures (deletions) and the client request rates for the various services. We denote as $\mathbf{A}(t) = (A_j^{v,s}(t))$ the number of VMs of type v for service s from provider j at slot t. Based on the SLAs and service profiling on the different VM types, the function $f(\cdot)$ determines the number of VMs per VM type required for handling client requests for each service within its SLA requirements. Also, $\mathbf{D}(t) = (D_{i,j}^{v,s}(t))$ is the number of VMs of type v for service s of provider j that terminate from server i at slot t.

At the beginning of each time slot t, the controller takes as input the instantiation of the random event $\boldsymbol{\omega}(t)$ and the edge cloud system state in terms of VMs deployed for the various services of the different MSOs, and determines the new VM deployments at the different nodes of the edge cloud. Let $\mathbf{a}(t)$ denote the four dimensional vector of control actions at slot t decided by the scheduling policy in effect. $a_{i,j}^{v,s}(t) \in \{0, 1, \ldots, A_j^{v,s}(t)\}$, $i \in \mathcal{N}$, $j \in \mathcal{P}$, $v \in \mathcal{V}$, $s \in \mathcal{S}$ represents the number of VMs of type v for service s of MSO j to be deployed at time slot t at edge-cloud node i.

The objectives of the IaaS controller for selecting the control action $\mathbf{a}(t)$ are the following:

Objective 1: For each VM of type v for service s that is created at the edge cloud, there is a rent price $w_{v,s}$ (i.e., an hourly rent price amortized per time epoch) that is paid to the edge cloud provider. The edge cloud provider would seek to allocate VMs to its physical machines, so as maximize its profit $W(t)$ per time slot t, given by:

$$W(t) = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{v \in \mathcal{V}} a_{i,j}^{v,s}(t) w_v \tag{3}$$

Over the time horizon, the cloud provider would seek to maximize the *expected cumulative profit* or equivalently the average expected profit over time, i.e.

$$\max \lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[W(\tau)] \,, \tag{4}$$

which can be written as a minimization problem as

$$\min \lim_{t \to \infty} -\frac{1}{t} \sum_{\tau=0}^{t-1} E[W(\tau)] \,. \tag{5}$$

Objective 2: For each request that is not served by a VM residing at the edge cloud, there is a penalty arising from the potential SLA violation for this service. We aim to minimize the *overall penalty* for the requests not handled at the edge cloud. We define the penalty function $P(t)$ as:

$$P(t) = \sum_{j \in \mathcal{P}} \sum_{s \in \mathcal{S}} \Bigg[r_j^s(t) - \sum_{v \in \mathcal{V}} \Bigg(\sum_{i \in \mathcal{N}} n_{i,j}^{v,s}(t) + a_{i,j}^{v,s}(t) - D_{i,j}^{v,s}(t) \Bigg) \cdot \xi_j^{v,s} \Bigg] \cdot \pi_j^s \,, \tag{6}$$

where π_j^s is the monetary penalty for not serving at the edge cloud a client request for service s of provider j.

Then, our objective for minimizing the *average expected penalty over time* is given by

$$\min \lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[P(\tau)] \,. \tag{7}$$

Objective 3: As a third objective, we seek to guarantee at least a minimum service at the edge cloud for all providers. This can be achieved by a *proportionally-fair* VM allocation among providers according to their relative SLA significance. To this end, we define a logarithmic utility function $u(\cdot)$ of the allocated resources x to each provider j as follows:

$$u_j(x) = \phi_j \log(x) \tag{8}$$

ϕ_j is the penalty for being unfair to provider $j \in \mathcal{P}$. We define the following function as an *unfairness metric*:

$$K(t) = -\sum_{j\in\mathcal{P}} \phi_j \cdot \log\left(\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}\sum_{v\in\mathcal{V}} n_{i,j}^{v,s}(t) + a_{i,j}^{v,s}(t) - D_{i,j}^{v,s}(t)\right) \quad (9)$$

Thus, regarding fairness our long run objective is

$$\min \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1} E[K(\tau)]\,. \quad (10)$$

Note that the logarithm in $K(t)$ can be approximated by a piecewise linear function, so that the objective in (10) is linear. Henceforth, such an approximation is assumed to be employed.

Constraints: The control actions $\mathbf{a}(t)$ to be selected by the IaaS controller at time slot t are constrained as follows. The VMs that will operate over some physical machine i at time slot t should not violate the physical capabilities $\mathbf{p}_i$ of the system for each resource. Let the vector $\mathbf{y}_i(t)$ represent the resource residuals at host i at time slot t, as an effect of the control action $\mathbf{a}(t)$ and the random event $\boldsymbol{\omega}(t)$ at time slot t (captured by the function $Y_i(\mathbf{a}(t), \boldsymbol{\omega}(t))$), given by:

$$\begin{aligned}\mathbf{y}_i(t) &= Y_i(\mathbf{a}(t), \boldsymbol{\omega}(t)) \\ &= \sum_{j}^{P}\sum_{v}^{V}\sum_{s}^{S}(n_{i,j}^{v,s}(t) + a_i^{v,s}(t) - D_{i,j}^{v,s}(t))\mathbf{m}_v \\ &\quad - \mathbf{p}_i\,.\end{aligned}$$

$n_{i,j}^{v}(t)$ is the number of VMs of type v that operate in physical machine i at any time slot t for provider j. Then, the feasibility constraints are given by:

$$\mathbf{y}_i(t) \leq \mathbf{0},\ \forall i \in \mathcal{N}\,. \quad (11)$$

Another constraint is related with the control actions: the VMs to be deployed at time slot t should be lower or equal to those requested, i.e.

$$\sum_{i\in\mathcal{N}} a_{i,j}^{s,v} \leq A_j^{s,v}, \forall s \in \mathcal{S}, v \in \mathcal{V}, j \in \mathcal{P}\,. \quad (12)$$

Now, the overall optimization problem becomes:

$$\textbf{Minimize: } \lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big(-E[W(\tau)] + E[P(\tau)] + E[K(\tau)]\Big) \quad (13)$$

$$\textbf{s.t. } (11), (12)$$

Therefore, the controller tries to select the VMs to be deployed for the various service providers over time, so as to maximize revenue (and edge-cloud utilization), deploy the most popular services, minimize the SLA violations and be fair among service providers at the same time.

TABLE I: Summary of Notation

Notation	Description
$\mathcal{N}$	Set of edge-cloud physical machines.
$\mathcal{V}$	Set of available VM types.
R	Physical capabilities of any physical server by means of $\{cpu, memory, storage, bandwidth\}$.
R_k	A vector $R_k = [m_k^j], j = 1, .., R$ with the physical capabilities R required per VM type.
P_i	The physical capabilities R of any physical server i by means of $R = \{cpu, memory, storage, bandwidth\}$.
$\mathcal{S}_j$	The set of VM-services for provider j.
r_j^s	The client request rate for service s that belongs to provider j.
$\tilde{r}_j^s$	The client request rate for service s that belongs to provider j and it is actually hosted at the edge cloud.

IV. Proposed Approach: Lyapunov optimization-Based VM Scheduling (LBVS)

The controller at each time slot selects the VMs to be deployed for the various service requests by the different MSOs. The edge cloud carries the effect of control decisions to the future, i.e., it is a *dynamic system*, while for the state there are some stability conditions. We exploit, Lyapunov optimization, and specifically the drift-plus-penalty technique [8], [9], [11], in order to achieve the long-run objective in problem (13). With this approach the minimization of the objective function depends only on the control actions and the random event (i.e., service arrivals and departures, client requests) at each slot and not on the whole system state, history or knowledge of distribution mean values. The alternative of calculating the optimal control actions over all time slots using dynamic programming would suffer from the curse of dimensionality and it would be impractical, as it would require knowledge of all random events over time in advance.

We assume that this problem is *feasible*, i.e., a control action that can satisfy all of the desired constraints exists. The virtual queues for the constraints are defined as follows:

$$\mathbf{Q}_i(t+1) = \max[\mathbf{Q}_i(t) + \mathbf{y}_i(t), \mathbf{0}],\ \forall i \in N \quad (14)$$

The Lyapunov function for the virtual queues is

$$L(t) = \frac{1}{2}\sum_{i\in\mathcal{N}} \mathbf{Q}_i^2(t)\,.$$

Using the drift-plus-penalty approach, we have

$$\begin{aligned}\Delta(t) + V(-W(t) + P(t) + K(t)) &\leq B \\ &+ V(-W(t) + P(t) + K(t)) + \sum_{i\in\mathcal{N}} \mathbf{Q}_i(t)\mathbf{y}_i(t)\,, \quad (15)\end{aligned}$$

where $\Delta(t) = L(t+1) - L(t)$ and B is a positive constant used to upper bound $\sum_{i=1}^{N} \mathbf{y}_i^2(t)$. The V parameter can be chosen adequately large, so as to ensure the time average of the objective is arbitrarily close to optimal, with a corresponding side-effect in the average virtual queue size. Thus, we can find almost optimal VM placement actions by greedily minimizing

at each slot t the following problem:

$$\textbf{Minimize}: V(-W(t) + P(t) + K(t)) + \sum_{i \in \mathcal{N}} \mathbf{Q}_i(t)\mathbf{y}_i(t) \tag{16}$$

$$\textbf{s.t.}\ (11), (12)$$

Due to Jensen inequality, it can be shown that an optimal solution to (13) can be achieved by solutions of the type $\mathbf{a}(t) = \mathbf{a}^*$, where $\mathbf{a}^*$ is a vector that solves the linear problem (16). Further, any time-averaged vector $\lim_{t\to\infty} \overline{\mathbf{a}(t)}$ corresponding to a solution of the time-averaged problem (13) must solve the linear problem (16). Therefore, the original linear optimization problem (13) can be solved by taking the time average of the decisions made when the drift-plus-penalty algorithm (16) is applied. The linear program (16) admits a worst-case polynomial-time algorithm with complexity $O((|\mathcal{V}| \cdot |\mathcal{S}| \cdot |\mathcal{N}| \cdot |\mathcal{P}|)^{3.5}L)$, where $|\mathcal{V}| \cdot |\mathcal{S}| \cdot |\mathcal{N}| \cdot |\mathcal{P}|$ is the number of problem variables that can be encoded in L input bits [12]. At each slot, a solver can be used to solve (16).

V. Evaluation

We evaluate the proposed scheduling approach by means of simulation experiments. The goals of the evaluation process were to verify the theoretical results, while also to demonstrate how the various edge-cloud and statistical parameters affect the performance of our approach. In addition, in order to demonstrate the compensation of optimality, we contrast the proposed scheduling policy with the well-known First Fit algorithm [13], i.e., place each VM into the first host where it fits, which is used as benchmark. First Fit algorithm is supposed to consider VMs according to two different orderings: i) Round-Robin (FF-RR), and ii) randomly (FF-Random). In FF-RR, small VMs are considered prior to medium ones, which are considered prior to large ones in a Round-Robin fashion among providers. In both cases of *edge-cloud parameters* and *statistical parameters*, a number of variables can be tuned, which could affect the effectiveness of the proposed scheduling policy in terms of convergence speed. For example, such edge-cloud parameters are the number of providers, the number of services or the number of host machines, and such statistical parameters, are the service request arrival rate distribution, the service lifetime distribution, the SLA-violation penalty and other weight factors. For brevity, we present here only indicative results out of extensive simulations.

A. Simulation Setup

For the evaluation of the proposed policies, we used a custom JAVA simulator and the IBM CPLEX optimization kit, which solves linear program (16) and determines VM deployments per time slot. For clarity of presentation of the system dynamics, we present the performance of our scheduling approach in a simple scenario with 2 providers. However, note that larger-scale experiments were performed with similar findings. Each provider sends requests for a single web service deployment over the edge cloud comprising a single physical machine. The same SLA contract is provided to both providers. This will be the basic setup for our simulation experiments unless stated otherwise. For this basic setup (see Table II), the service requests follow a Poisson distribution for both providers, whereas provider 2 has a higher service deployment request rate. Furthermore, in order to estimate the expected client request load per service in runtime, a simple moving average with window size 3 is employed based on prior load measurements. Each simulation run lasts 100 time slots and we average our results over 100 runs.

TABLE II: Basic Setup - Simulation Parameters

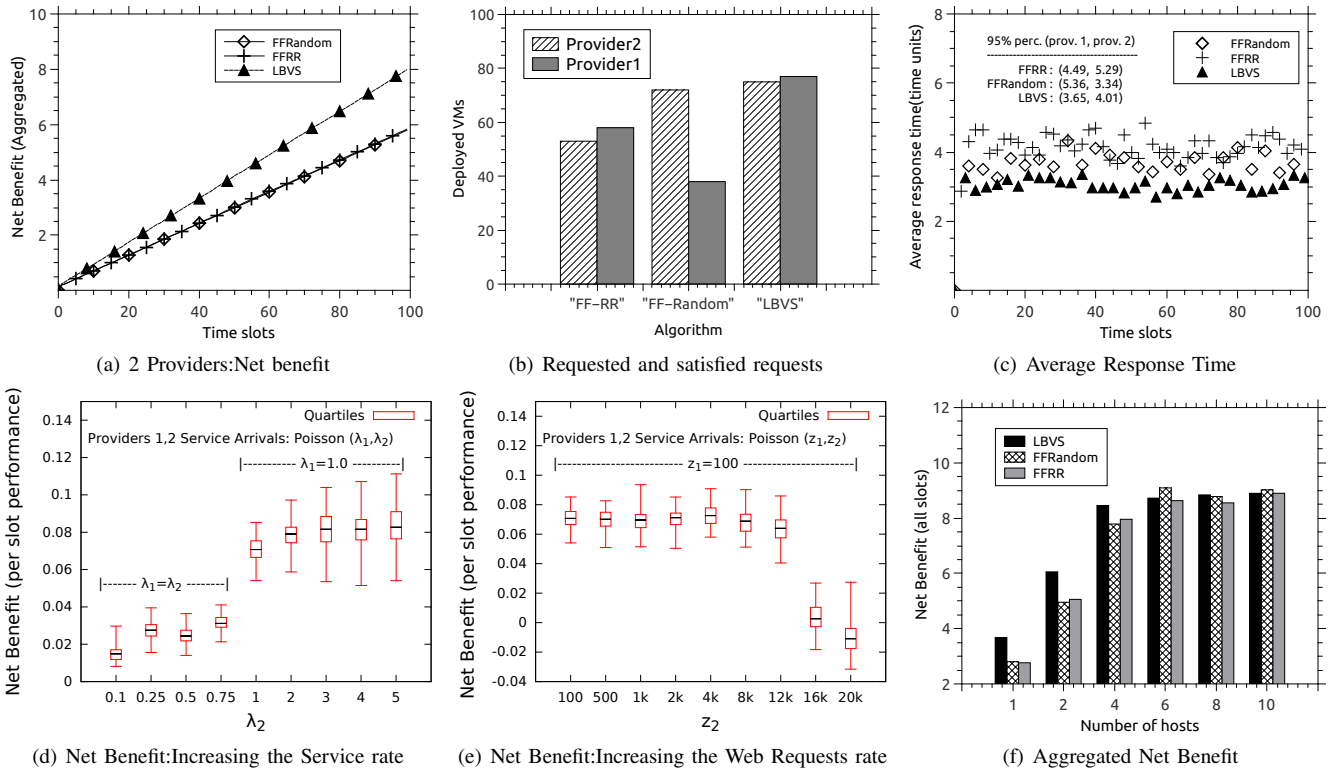
Resources	Host	VM (small,mdm,lrg)
CPU	8	(1,2,4)
Memory	100000	(1024,2024,4024)
Storage	100	(10,20,40)
Bandwidth	10	(1,1,1)
Provider Config.	**Provider 1**	**Provider 2**
Service arrival	Poisson, λ=1	Poisson, λ=2
Web Request arrival	Poisson, z=100	Poisson, z=100
Service lifetime	Exponential, μ=2	Exponential, μ=2
Local Cloud Resp. time	Exponential, μ=1	Exponential, μ=1
Rem. Cloud Resp. time	Exponential, μ=10	Exponential, μ=10
Client Requests within SLA	ξ^{small}=5000, ξ^{mdm}=10000, ξ^{lrg}=20000	ξ^{small}=5000, ξ^{mdm}=10000, ξ^{lrg}=20000
VM Rent (small, mdm, lrg)	0.026, 0.052, 0.104	0.026, 0.052, 0.104
SLA-violation Penalty	1	1
Unfairness Weight	1	1

B. Simulation Results

In Fig. 4(a), we present a performance comparison of the three scheduling policies considered in terms of cumulative net benefit in the basic scenario. Evidently, even in this simple case where a single service is requested by each provider, the LBVS approach outperforms the other two policies, since it is more capable to adapt on service-deployment load differences (provider 2 has increased service deployment rate). Therefore, LBVS results to better utilization of the physical resources, as compared to both variants of the First Fit scheduling policy.

This result is also explained by Fig. 4(b) presenting the number of deployed VMs per scheduling policy. Although the total number of VM deployments can reach a saturation point due to capacity constraints, LBVS approaches closer to the theoretically optimal bound of VM deployments without any knowledge of the statistical properties of the random variables (i.e., service-deployment requests, service lifetime, load of client requests) of the problem. Moreover, note that the LBVS approach is fairer for the providers, as compared to the benchmarking scheduling policies. The better utilization of the Edge-Cloud by the LBVS scheduling policy as compared to the benchmarking ones is also reflected to the response time of client requests in Fig. 4(c). As depicted therein, LBVS achieves lower response time than rival scheduling policies.

Then, we investigate the effects of increasing the service load (and consequently the VM load), while keeping fixed the client-request load for the service. As shown in Fig. 4(d), keeping the VM-deployment rate of provider 1 fixed (λ_1=1 VM request/slot) and increasing the VM-deployment rate of



(a) 2 Providers:Net benefit (b) Requested and satisfied requests (c) Average Response Time

(d) Net Benefit:Increasing the Service rate (e) Net Benefit:Increasing the Web Requests rate (f) Aggregated Net Benefit

Fig. 4: Evaluation results

provider 2, a saturation point is reached at point $\lambda_1 = \lambda_2 = 1$, after which the net benefit of the Edge-IaaS cannot be increased. This saturation point means that resource capacity limits of the Edge-Could have been reached. Note that, in this case, after the capacity limit of the Edge-Cloud has been reached, the net benefit of the Edge-IaaS that equals (rent from VM deployments)-(penalty for SLA violation)-(penalty for being unfair) (i.e., eq.(3)-eq.(6)-eq.(9)) remains constant. This is because, no additional rents can be obtained, while penalty factors remain unaltered, since web-service client requests remain fixed. This is not the case when the client-request load for the web service increases, as depicted in Fig. 4(e). Specifically, we examine the case where the VM/service load is fixed, while we increase the web load for both providers. As depicted in Fig. 4(e), after the total number of client requests that can be served within the SLA by the VMs deployed at the edge cloud is exceeded, the SLA-violation penalty increases, while the rent from deployed VMs and the unfairness penalty remain stable. The concluding remark for the cases presented in Figs.4(d) and 4(e) is that a very careful SLA definition is necessary when strict performance guarantees are to be provided to services. As experimentally shown, even with a close-to optimal scheduling policy, such as LBVS, penalties arising from SLA violations may be critical and may even nullify the profit from deploying VMs in the edge cloud.

Finally, in Fig. 4(f), we present the total cumulative net benefit for both providers over time as the number of nodes of the edge cloud increases. As expected, when the edge cloud has abundance of resources, every scheduling policy achieves a high net benefit for the Edge-IaaS, as all service/VM deployment requests are satisfied in this case. However, the more limited the edge-cloud resources, the higher the performance gap achieved by the LBVS scheduling policy over the benchmark ones.

VI. Related Work

There has been much prior work related to VM scheduling. Here, we overview some important ones that are more relevant to our approach and closer to the service-based VM allocation notion that we consider. Lyapunov optimization technique is described in numerous works like [7], [8], [9] and is being used for minimization of energy consumption in the data center [7] or power control in wireless networks [8], [9].

An approach with similar objectives to ours was proposed in [4]. The authors propose the "CloudScale" mechanism, which, residing at a physical node, performs VM resource consumption monitoring and prediction, in order to adjust resource allocation to VMs for preventing SLO violations. It employs FFT for discovering repeating patterns and predicting demand, and it employs proactive resource padding for correcting resource-allocation under-estimation errors based on their weighted moving average. If resource demand under-estimation occurs for a VM, it re-actively adjusts the VM resources based on a scaling factor per time slot. When the total forecasted resource demand for VMs exceed the capacity of the host, CloudScale scales down resource allocations to local VMs or migrates VMs. However, since CloudScale mechanisms at different hosts run independently, SLO violations can also occur due to collisions during VM migrations. This situation

is not addressed in [4]. In [14], the past demand behavior of a VM at a host is used for future placement. The VMs are placed so as to minimize future unsatisfied load for the various VMs based on observations on the previous TM slots.

In [15], an HPC-aware VM scheduler is proposed. The approach employs multi-dimensional bin packing and places VMs to homogeneous hardware resources in a topology-aware manner, so as HPC applications with tight coupling to have low communication latency. Also, in [15], the authors calculate the VM cross-interference by application characterization via cache intensiveness and communication needs. In [5], the authors schedule VMs in the cloud using multidimensional bin packing with stochastic and deterministic resource demands. The approach aims to maximize the minimum utilization of all servers and satisfy SLAs with stochastic guarantees. The demand for a stochastic resource (e.g., network I/O) is calculated based on its distribution and a probability threshold that the capacity of the server for the stochastic resource is violated. Multi-dimensional bin packing is solved by means of First Fit Decreasing (FFD) and Dominant Resource First (DRF) algorithms. In [16], the authors try to find fast solutions to the NP-hard bin packing problem VM placement based on GPU-based acceleration. However, their approach performs static VM resource allocation and placement to servers and it does not account for VM resource demand fluctuations. In [17], physical machines are grouped into cohorts based on migration capabilities, namely from live migration, migration through suspend/resume, no migration at all. Then, scores are assigned to cohorts according to their resource availability and their capacity constraints and best-ranked cohorts are employed for VM placement. In [18], the VM physical machine cost (PM-cost) and the inter-VM communication cost (N-cost) are jointly considered, so as to minimize the overall scheduling cost. Three models of the N-cost are considered based on the number of non-collocated VMs for the same tenant and the VM sizes. For different requests for VMs from the various tenants, their algorithm aims to minimize N-cost, thus requests that ask for more VMs are placed first. For overall cost minimization, they propose an approximation algorithm based on binary search for the number of PMs that achieve a better trade-off between N-cost and PM-cost. In [6], VMs are supposed to be allocated for two kinds of jobs with different SLA requirements, namely (a) the batch non-interactive computationally-intensive (HPC) jobs that have to be completed within a deadline and (b) the transactional (web) jobs that have to execute with a certain response time. In the initial allocation transactional applications are allocated resources equal to their peak demand. For non-interactive jobs, the scheduler tries to allocate slack of resources remaining at a host that is running a transactional application. Resources allocated to VMs are adjusted according to forecasted demand based on Artificial Neural Network (ANN).

VII. Conclusions

In this paper, we investigated the problem of service/VM scheduling in environments with limited physical resources, like the edge cloud, where multiple service providers and service operators need to deploy time-critical services over a shared infrastructure. We proposed an SLA-driven VM scheduling approach that maximizes the revenue of the Edge-IaaS and minimizes SLA violations, while being fair to the various service providers. The problem was solved by a Lyapunov optimization framework and the performance our proposed scheduling approach was experimentally validated by means of simulation experiments. As a future work, we plan to enhance our model to consider VM migrations and more advanced load-forecasting methods. We also intend to implement our LBVS policy in a real testbed and evaluate it in RAN-sharing concepts.

Acknowledgment

This work has been supported by the 5G-PPP project "Coordinated Control and Spectrum Management for 5G Heterogeneous Radio Access Networks" (COHERENT).

References

[1] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-Edge Computing Introductory Technical White Paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[2] N. Alliance, "5G white paper," *Next Generation Mobile Networks, White paper*, 2015.

[3] ETSI, "Mobile Edge Computing (MEC): Framework and Reference Architecture," *GS MEC 003 V1.1.1*, 2016.

[4] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *ACM SOCC*, 2011.

[5] H. Jin, D. Pan, J. Xu, and N. Pissinou, "Efficient VM placement with multiple deterministic and stochastic resources in data centers," in *IEEE GLOBECOM*, 2012.

[6] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter," *Journal of Network and Computer Applications*, vol. 45, pp. 108–120, 2014.

[7] R. Urgaonkar, U. C. Kozat, K. Igarashi, and M. J. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *IEEE NOMS*, 2010.

[8] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[9] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

[10] G. B. Dantzig, "Discrete-variable extremum problems," *Operations Research*, vol. 5, no. 2, pp. 266–288, 1957.

[11] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *Trans. on Automatic Control, IEEE*, vol. 37, no. 12, pp. 1936–1948, 1992.

[12] N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.

[13] B. Xia and Z. Tan, "Tighter bounds of the first fit algorithm for the bin-packing problem," *Discrete Applied Mathematics*, vol. 158, no. 15, pp. 1668–1675, 2010.

[14] N. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti, "VM Placement Strategies for Cloud Scenarios," in *IEEE CLOUD*, 2012.

[15] A. Gupta, L. Kale, D. Milojicic, P. Faraboschi, and S. Balle, "HPC-Aware VM Placement in Infrastructure Clouds," in *Cloud Engineering (IC2E), IEEE*, 2013.

[16] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *ACM SOCC*, 2012.

[17] K. Tsakalozos, M. Roussopoulos, and A. Delis, "VM Placement in non-Homogeneous IaaS-Clouds," in *Service-Oriented Computing*, 2011.

[18] X. Li, J. Wu, S. Tang, and S. Lu, "Let's stay together: Towards traffic aware virtual machine placement in data centers," in *IEEE INFOCOM*, 2014.