Cost and Availability Aware Resource Allocation and Virtual Function Placement for CDNaaS Provision

Louiza Yala, Pantelis A. Frangoudis, Giorgio Lucarelli, and Adlen Ksentini, *Senior Member, IEEE*

***Abstract*—We address the fundamental tradeoff between deployment cost and service availability in the context of on-demand content delivery service provision over a telecom operator's network functions virtualization infrastructure. In particular, given a specific set of preferences and constraints with respect to deployment cost, availability and computing resource capacity, we provide polynomial-time heuristics for the problem of jointly deriving an appropriate assignment of computing resources to a set of virtual instances and the placement of the latter in a subset of the available physical hosts. We capture the conflicting criteria of service availability and deployment cost by proposing a multi-objective optimization problem formulation. Our algorithms are experimentally shown to outperform state-of-the-art solutions in terms of both execution time and optimality, while providing the system operator with the necessary flexibility to balance between conflicting objectives and reflect the relevant preferences of the customer in the produced solutions.**

***Index Terms*—Content delivery networks, cloud computing, network functions virtualization, resource allocation, service availability, VNF placement.**

I. Introduction

RECENT studies in networking and cloud computing focus on virtualizing network functions with the aim of providing highly scalable and flexible service deployment, cost reductions, and improved end-user experience. The ETSI has recently proposed a Network Functions Virtualization Management and Orchestration framework (NFV-MANO) [1], which specifies a set of architectural components and interfaces to realize the NFV vision. Among the many use cases envisioned in such an environment is the provision of a virtualized Content Delivery Network (vCDN) service [2, Use Case #8].

Manuscript received April 15, 2018; revised August 12, 2018; accepted October 1, 2018. Date of publication October 8, 2018; date of current version December 10, 2018. This work was supported in part by the French FUI-18 DVD2C project and by the European Union's Horizon 2020 research and innovation program under the 5G-Transformer project (grant no. 761536). The associate editor coordinating the review of this paper and approving it for publication was G. Schembra. *(Corresponding author: Pantelis A. Frangoudis.)*

L. Yala is with IRISA/University of Rennes 1, 35042 Rennes, France (e-mail: louiza.yala@irisa.fr).

P. A. Frangoudis and A. Ksentini are with the Communication Systems Department, EURECOM, 06904 Sophia Antipolis, France (e-mail: pantelis.frangoudis@eurecom.fr; adlen.ksentini@eurecom.fr).

G. Lucarelli was with CNRS, INRIA, LIG, University Grenoble Alpes, 38000 Grenoble, France. He is now with LCOMS, University of Lorraine, 57000 Metz, France (e-mail: giorgio.lucarelli@univ-lorraine.fr).

The interactions between CDN providers and network operators [3] have received considerable attention. Many content delivery models involve different degrees of cooperation between stakeholders [4]. The *telco CDN* is one of the studied models, where a telecom operator installs data centers at points of presence in its network and offers a CDN service to content providers. NFV facilitates the deployment of such a service. The operator moves from deploying dedicated hardware for hosting CDN components, such as content origin servers, caches, load balancers, and DNS resolvers, to leasing its telco cloud resources for launching virtual instances (Virtual Machines (VMs) or other containers) of the above components on demand, allocating resources intelligently at the optimal locations in its network, and scaling them to match dynamic workloads.

In our prior work [5], we presented an architecture for CDN-as-a-Service (CDNaaS) provision over a network operator's cloud, with a design following the NFV-MANO spirit. Our architecture allows content providers to request the dynamic instantiation of a full vCDN using RESTful northbound interfaces, expressing their expected end-user demand per region, as well as Quality of Experience (QoE) and service availability requirements. At the heart of this system, service orchestration components are responsible for taking all necessary steps for run-time management of the vCDN instance on the underlying NFV Infrastructure (NFVI) of the operator, abstracting internal details and offering the customer (content provider) only the necessary entry points to the vCDN, e.g., in order to infuse content or terminate the service.

One distinctive characteristic of our design is that it allows for the application of a multitude of service-tailored dimensioning, resource allocation, and resource management algorithms in a plugin fashion. This article focuses on these aspects in particular. A vCDN deployment request generally involves solving the following problems: (i) Deciding on the necessary amount of computing (and other) resources to dedicate in order to efficiently respond to user requests. (ii) Deriving an appropriate placement of virtual CDN service components on the underlying NFVI, with the aim ideally of minimizing cost and offering availability guarantees. Trying to achieve these two objectives can be compromising, due to their conflicting nature; employing more virtual or physical resources to improve on service availability via redundancy and fault tolerance naturally increases management (e.g., power consumption) cost. It should be noted that the importance of

resilience and availability for NFV has been acknowledged by the ETSI NFV Industry Specification Group, which has published specific requirements and guidelines, identifying, among others, the cost-availability tradeoff [6].

We have provided solutions for the first problem in our prior work [7], where the main objective was to combine customer-provided demand dimensioning information (e.g., target number of users/video streams per region) with network and compute infrastructure awareness for optimal resource allocation. For that, we focused on a video content delivery service; we experimentally quantified the relationship between video QoE and service workload, and used this information to optimally decide on the amount of CPU resources to allocate to satisfy customer QoE requirements while minimizing the cost for the operator. With this as a starting point, this article provides solutions to the second problem, namely the distribution of the necessary virtual computing resources to a number of virtual service components and the appropriate placement of the latter in the operator's NFVI. We make the following contributions:

- We capture the conflicting objectives of service availability and deployment cost by proposing a multi-objective optimization formulation for the problem of joint compute resource allocation and VM placement, which we initially introduced in [8].
- By relaxing some of the problem's assumptions, we provide polynomial-time heuristic algorithms to solve it and show experimentally that the derived solutions are close to the optimal, while at the same time being significantly faster than an exact algorithm provided by a commercial solver.
- We show our approach to outperform two baseline schemes (random placement and first-fit), as well as a genetic algorithm inspired from the state of the art [9] which we have devised as an alternative solution.

This article is organized as follows: In Section II we review the state of the art. In Section III we provide an overview of our architecture for CDNaaS provision over a telco cloud, in the context of which we put our proposed algorithms. Furthermore, we provide an overview of our measurement-driven methodology for determining the amount of CPU resources necessary to satisfy specific QoE levels for a CDN service, presented in our prior work [7], which is the input to the algorithms presented in this article. In Section IV we propose a model for joint vCPU-to-VM allocation and VM placement, and propose a heuristic solution for a relaxed version of it in Section V. Furthermore, we study an alternative way to tackle our problem by proposing an adaptation of a genetic algorithm from the state of the art in Section VI. We present experimental results on the performance of our scheme compared with heuristics widely used in the literature, the aforementioned genetic algorithm, and an exact solution in Section VII and conclude the article in Section VIII.

II. Related Work

The placement of virtual instances to physical hosts is central in NFV in particular, and cloud computing in general. The main objective is to find a suitable set of physical machines (PMs) with enough capacity to host the virtual instances with specific resources allocated to each one of them. In NFV, these instances host Virtual Network Functions (VNFs), and are implemented as VMs or other container technologies. A suitable VNF placement is mostly motivated by maximizing resource utilization or overall performance, while minimizing cost in terms of energy consumption, network traffic or penalties associated with SLA violations under various constraints [10]–[12].

Mann [13] reviews different VM placement models and algorithms, citing, among different formulations, the problem of packing the VMs into a minimal number of PMs, considering PM capacities and the load of VMs. This reduces to the bin packing problem which is NP-hard. Various heuristics to solve it in this context exist, such as first-fit, where the items (VMs) are placed in the first suitable bin (PM) [14], [15].

Li and Qian [16] survey different network function orchestration frameworks. In particular the authors compare different network function placement strategies and highlight the positive and negative points of different approaches of VNF placement.

The common ground in the way various flavors of the VM placement problem are tackled is the need to provide heuristics to problems with typically high computational complexity. Minimizing the number of physical hosts to place VMs for cost and performance optimizations, a procedure known as VM consolidation, may have an adverse effect on service resilience and availability in the face of PM failures. Contrary to the aforementioned works, service availability is our special focus.

Mijumbi *et al.* [17] propose an evaluation of resource allocation algorithms considering parameters such as successful service mappings, total service processing times, revenue or cost under varying network conditions. In particular, they define cost as the total amount of physical resources that are used by a given mapping and scheduling. The difference between revenue and cost is that the revenue only consists of the processing time of the functions, while the cost involves time gaps that are left unused due to functions waiting for their assignment. On the other hand, Bari *et al.* [18] propose an optimization problem formulated as an Integer Linear Program (ILP) referred to as the VNF orchestration problem (VNF-OP), which consists in minimizing the cost (deployment cost, energy cost, and cost of forwarding traffic), penalties for Service Level Objective (SLO) violations, and resource fragmentation. The constraints taken into account are related with server/link capacities and service chaining. In our work we consider cost as the fixed managerial overhead of operating a virtual machine (resp. physical machine), which is not a function of its workload.

Integer Linear Programming (ILP) has been a popular modeling technique for VM allocation problems. However, experimental results from Rankothge *et al.* [19] show that using ILP to find an optimal configuration can have a very long execution time even for a small number of network functions. The authors thus studied approximations by means of finding the best fitted solution according to a Genetic Algorithm (GA) model of the problem. In particular, they proposed a Genetic

Programming-based approach to solve the VM allocation and network management problem, exploring a fixed amount of generations. Although their GA may not provide the optimal solution, it can compute configurations orders of magnitude faster than ILP. GAs are a part of evolutionary computing and were introduced as a computational analogy of adaptive systems [20, p. 64–75].

The relevant state of the art also includes the GA of Xu and Fortes [9], which aims to solve VM placement formulated as a multi-objective optimization problem of simultaneously minimizing the total resource wastage, power consumption and maximum thermal dissipation. With this work as a starting point, in Section VI we propose a GA tailored to our problem settings. The heuristic algorithms we present in this article, however, are shown to outperform our GA both in terms of approximating the optimal solution and execution time.

Yang *et al.* [21] focus on reliable VM placement, and, in a similar spirit to our case, model service availability as the probability that a subset of the requested VMs is operational. However, they address a different VM placement problem, which, among others, does not consider the distribution of the resources (in our case vCPUs, in their case storage) to be committed, assuming fixed VM resource specifications, and does not take into account VM-level failures. Ecarot *et al.* [22] present a method for cloud resource allocation which aims to satisfy both consumers' (or end-users') and providers' interests. Their model is an ILP which minimizes the costs and service unavailability, the latter corresponding to the penalties for quality of service degradations or violations by the providers of a service, which are in turn associated with excessive service workload. This is contrary to our approach, where availability is defined as the probability that a virtualized service is accessible. To deal with the complexity of the problem, the authors apply evolutionary algorithms. Others, such as Casazza *et al.* [23], propose to guarantee the availability of a service by replicating VNFs across multiple servers. They apply a probabilistic metric for availability but, contrary to our work, they aim at maximizing the minimum service availability considering the latter as their sole objective under server capacity and other constraints.

Qu *et al.* [24] focus on the problem of VNF chaining and aim to minimize the network-wide communication bandwidth for the operation of VNF chains under service reliability constraints. Their model does not consider failures at the VM level in its reliability functions and their focus is rather on chain-specific issues such as routing and VNF ordering, which constrain VNF-to-host placement. Furthermore, their model does not include CPU capacity constraints.

Regarding cost modeling, Callau-Zori *et al.* [25] experimentally quantify how the number of VMs deployed impacts both energy cost and performance. This work verifies our model assumption that cost is linear to the number of VMs and PMs utilized.

Ouarnoughi *et al.* [26], on the other hand, focus on storage cost. They propose a detailed cost model for IaaS infrastructures which takes into account at the same time various parameters, such as VM execution and storage migration, system storage performance, power and wear out. They also consider cloud Service Level Agreement (SLA) violations and the associated penalties. Our cost model is more abstract, encompassing all related sub-costs in linear functions of the number of PMs and VMs utilized. Also, it is not considering the operational cost due to service workload, since this cost is assumed proportional to the workload and in any case independent on the number of PMs or VMs deployed, thus not influencing the selection of an appropriate virtual function placement.

TABLE I
STATE OF THE ART CLASSIFICATION ACCORDING TO THE CRITERIA CONSIDERED

Network operation/management and other costs	[22], [19], [18]
Energy consumption	[10], [11], [25], [9]
Availability	[24], [22], [23]
Failure recovery	[27]
SLA violation	[12], [26]

A relevant aspect is fault recovery to maintain high cloud service availability. To tackle this issue Israel and Raz [27] propose approximation algorithms with performance guarantees and heuristics. This topic is outside the scope of this article.

A classification of existing papers dealing with the VNF placement problem proposed in the state of the art is presented in Table I, according to the different optimization criteria used in each model.

As can be seen in Table I, most of the works in the relevant literature focus only on a single objective, knowing in advance the number of VNF instances to place and having as an objective to place these virtual instances on physical hosts. Rare are those works which treat the subjects of availability and cost jointly, and there lies the distinctive characteristic of our approach. Furthermore, we should note that in the context of CDNaaS provision, our design tackles the problem of compute resource allocation for a video service and virtual instance placement in a holistic manner, although this article mainly focuses on the latter, namely the joint vCPU to VM assignment and VM-to-PM placement.

III. A CDNAAS ARCHITECTURE FOR THE TELCO CLOUD

A. Design

In our prior work [5] we presented an architecture which offers the flexibility to a telecom operator to lease its CDN infrastructure in a dynamic manner, offering a virtual CDN (vCDN) service that can be deployed on demand over the operator's private cloud. We highlighted the fact that our design offers a flexible sharing by allowing the customer to, for example, use the leased infrastructure to respond to predicted traffic surges at particular regions, enjoying the network operator's regional presence. From the network provider side, this concept allows more efficient use of its infrastructure resources, compared to a less dynamic resource reservation model with static allocation of data center resources to customers.

Our CDNaaS architecture (Fig. 1) involves various functional blocks which communicate via well-specified interfaces.

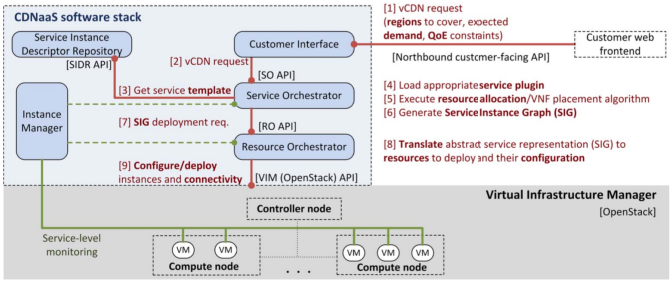

Fig. 1. CDNaaS architectural components [5].

This decouples their operation from any physical location, allowing the CDNaaS provider to execute any block autonomously as a virtual function over its own, or any, cloud infrastructure. Notably, our design is in line with the ETSI NFV-MANO specification [1], with its functional blocks and interfaces mapping to MANO components.

Via the Customer Interface Manager component, our system features a northbound RESTful API through which customers (content providers) can request to deploy a virtual CDN over the telco cloud. This API offers the customer a way to specify details on the expected end user demand for its service per region, and, importantly, a target QoE level for the end users of the content delivery service.

This information is critical for service deployment, a task which is coordinated by the Service Orchestrator (SO) component. In particular, the SO is responsible for deriving an appropriate compute resource allocation and VNF placement, taking into account the customer requirements (demand and QoE) included in the service request and its operational capacity. With flexibility in mind, our design allows for the application of various such schemes in a pluggable manner. This article targets this aspect in particular: While in [5] we focused on the infrastructure support, here we provide concrete, low-complexity algorithmic solutions for the problem of compute resource allocation and VNF placement for a vCDN instance.

The output of the above process is represented as a Service Instance Graph (SIG), which maps VNF instances (VMs) and the respective resources allocated to each one of them to physical nodes. The SIG is then handed to the Virtual Infrastructure Manager (VIM) for deployment on the underlying cloud substrate. Since OpenStack [28] is the de facto VIM software solution, we opted for its use in our system implementation.

B. Virtualized CDN Service-Level Features

Although our architecture is fairly generic and could support heterogeneous types of services,[1] we focus on the specifics of a video content delivery service. Shifting our attention from infrastructure- to service-level features, we present some specific characteristics and assumptions regarding the operation of the vCDN application.

[1]For example, apart from supporting various different CDN flavors, we have successfully used our scheme to orchestrate the deployment of a virtualized Evolved Packet Core network, in a 4G mobile network context.

The vCDN delivers video content over HTTP via a number of caches which operate as streaming servers. Caches are implemented as VNFs. An end-user request for a video item is redirected to a nearby cache using DNS geolocation techniques (other options are also possible). Each regional cache is composed of a set of VMs with identical functionality (HTTP servers caching and streaming video) and user requests are transparently balanced among them using an HTTP load balancer.

The CDNaaS operator has data centers in a number of regions, on which vCDN instances for video distribution can be deployed. The dimensioning and placement algorithms are to be executed per region included in the customer request. This is because the target of the operator is to deliver content to each region's end users from the local data center. We assume that vCDN traffic is handled exclusively by the vCDN instance components assigned to a specific region and thus resources are allocated in a region-local manner. Issues regarding redirecting user traffic across regional data centers are outside the scope of this work. For each region, a sufficient number of virtual CPUs needs to be allocated for cache (streaming) servers to cope with the expected service demand and QoE constraints. The vCPUs are distributed to a number of VMs which are identical from a functional perspective for reasons of fault tolerance, but also because it may not be possible to consolidate them in a single VM, due to potential capacity limitations of the underlying physical host. From a performance viewpoint, we assume that the number of VM replicas used to deliver a service does not have any effect; only the number of vCPUs allocated to handle the service workload matters.[2]

In this work, we pay a particular attention to the fault tolerance of the vCDN service. By design, and not taking into account other service components such as load balancers and DNS servers, the video service is considered available in a region if at least one VM hosting a cache is accessible to users. This is due to the load balancers which transparently redirect user requests to available local cache instances. It should be noted that service availability defined as such does not guarantee that the required QoE levels are met in the event of failures, since the latter depend on the number of compute resources available for responding to user demand. If a VM fails and appropriate repair activities are not in place, the CPU resources allocated to this VM become unavailable. However, with appropriate service monitoring and management schemes, this issue could be adequately addressed: The released CPU resources could be automatically and transparently reallocated to a running VM on the same host, if such one already exists (scaling up), VMs on other physical hosts covering the same region could be scaled up if there is available CPU capacity in their hosts, or, as a last resort, the failing VM could be relaunched on the same host. Such self-healing/repairing techniques are outside the scope of this article.

[2]In practice, using a load balancer to distribute user requests for content can have a performance effect (e.g., increased latency) compared to serving requests directly, without the intervention of the load balancer. However, we consider these performance effects negligible in this study.

C. Resource Allocation

This procedure is composed of two main phases. First, a resource dimensioning algorithm which decides on the optimal allocation of vCPU resources per region needs to be executed, aiming to satisfy the constraints dictated by the customer request and the operator's capacity for a given service demand. For a video streaming vCDN service, these can be expressed in terms of a maximum number of concurrent users that will be accessing the video service per region, respecting the minimum QoE constraint set by the customer, which takes the form of a minimum Mean Opinion Score (MOS) for specific video characteristics (e.g., high definition video).

In our prior work [7], following extensive testbed measurements, we derived an empirical model of video QoE as a function of the compute resources utilized and the workload in terms of parallel video streaming sessions. Based on this model, we proposed a mechanism for the CDNaaS provider to optimize the amount of compute resources to allocate to guarantee the desired video QoE levels, and demonstrated how such informed resource allocation decisions offer savings on operator costs while improving on user experience. In particular, this mechanism calculates the minimum number of vCPUs needed to serve a given customer-specified demand under capacity (operator) and quality (customer) constraints.

The second phase, which is the main focus of this article, uses as input the amount of compute resources (number of vCPUs) calculated in the first phase and derives an appropriate assignment of them over a number of VMs to be deployed on a subset of the available telco cloud physical hosts. How exactly these resources are allocated and how the respective virtual instances are placed is a matter of addressing the trade-off between availability and cost: Splitting resources across instances and distributing the latter across multiple physical hosts can result in higher service availability, but also increased operational cost (e.g., power consumption) for the operator, which may be reflected in the service price. In this work we respond to the latter by proposing a multi-objective optimization formulation for the joint CPU resource allocation and virtual instance placement problem, where customer preferences are translated to specific weightings for the two conflicting objectives. Note that the two resource allocation phases are independent. The model and algorithms presented in this article could be directly applied in conjunction with any algorithm for deriving the number of vCPUs necessary for a service request.

We should note that our model and algorithms, detailed in Sections IV and V, have been introduced in our prior work [8]. This article extends this work by putting them in context of our CDNaaS design and presenting a comparison with an adaptation of a state-of-the-art genetic algorithm which we also implemented, beyond the random and first-fit placement methods which are typically used in the literature as benchmarks. Importantly, in this article we also delve into computational aspects; we discuss the problem's complexity and quantify the performance of our algorithms in terms of execution time vs. solution quality. For the latter, we have implemented our model using the CPLEX solver to derive exact solutions and compare them with our heuristic ones.

IV. A Model for Joint vCPU-to-VM Allocation and VM Placement

A. Preliminaries

We aim to assign p vCPUs, the output of the first phase of the resource allocation procedure, to a number of up to p virtual machines (VMs), and the placement of the latter in (a subset of) the available m physical machines (PMs) of a regional data center, aiming to minimize the deployment cost and to maximize service availability, while respecting PM vCPU capacity constraints. The outcome of this process is a matrix $X = (x_{ij})$ with $i \in [1, p]$ and $j \in [1, m]$, where x_{ij} denotes the number of vCPUs assigned to VM i hosted in PM j. The upper bound for i is the number of vCPUs to assign (since our unit of processing is a vCPU, we cannot have more VMs than the number of vCPUs to assign). The calculation of the optimal assignment should respect physical capacity, cost, and availability constraints.

B. Cost Model

We consider that the deployment of a VM comes with a fixed management overhead which is not a function of its workload. For example, this cost can account for the energy consumed for booting the VM or for operating other system- or service-level components (e.g., operating system). We further assume that for each PM which hosts service instances, there is a fixed overhead which is not a function of the PM workload nor the number of VMs hosted by it (e.g., energy cost for keeping the physical machine in an operating state, overhead of various system-level components). We model the above costs as linear functions of the number of VMs and PMs utilized by a service deployment, which is in line with the experimental observations of Callau-Zori *et al.* [25].

In matrix X, the number of non-zero elements represents the number of VMs deployed. Therefore, the cost of an assignment X at the VM level is given by

$$C_V(X) = e_V \sum_{i=1}^{p} \sum_{j=1}^{m} \big(x_{ij} > 0\big), \tag{1}$$

where e_V is the fixed cost per deployed VM. In a similar spirit, the cost of an assignment X at the PM level is determined by the number of PMs that host at least one VM. This corresponds to the number of columns in X that contain at least one non-zero element. This cost is thus given by

$$C_P(X) = e_P \sum_{j=1}^{m} \left(\sum_{i=1}^{p} x_{ij} > 0 \right), \tag{2}$$

where e_P is the fixed cost per used PM, and the overall cost of an assignment follows:

$$C(X) = C_V(X) + C_P(X). \tag{3}$$

C. Availability Model

We define service availability as the ability of the system to offer at least a minimal service, i.e., to have, at any time, at least one VM accessible, which implies that at least one PM should be up to host the respective VM(s).

We make the following assumptions:

- A VM *i* can fail with probability $q_i^{(V)}$, independently of the other VMs and PMs, and irrespectively of the load imposed on the VM.
- Each PM *j* can fail with probability $q_j^{(P)}$, independently of the other PMs or the load imposed on it. The above probabilities are assumed to be known to the operator as a result of measurement studies, prior experience, or other historical information. (The same applies to VM failure probabilities.)
- If a PM fails, all VMs deployed on top of it are assumed to fail because of that.

Therefore, a VM may become inaccessible either because it fails or because the PM that hosts it fails. VM failures can be correlated due to their dependence on the underlying PMs. Based on this, we define a *correlated group* of VMs as the VMs which are executed on the same PM. For a correlated group to be available, the following conditions should hold:

- The PM is up, and
- At least one of the VMs deployed on the PM does not fail.

The probability that a correlated group deployed on PM *j* is available is thus given by:

$$a_j = \left(1 - q_j^{(P)}\right)\left(1 - \prod_{i \in [1,p] | x_{ij} > 0} q_i^{(V)}\right) \tag{4}$$

For a vCDN service deployment to be available, at least one correlated group should be available. Since correlated groups fail independently, the probability that a vCDN service deployment is available is given by

$$\begin{aligned} & A(X) \\ & = 1 - Pr\{\text{All correlated groups fail}\} \\ & = 1 - \prod_{j \in [1,m] | \Sigma_{i=1}^{p} x_{ij} > 0} (1 - a_j) \\ & = 1 - \prod_{j \in [1,m] | \Sigma_{i=1}^{p} x_{ij} > 0} \left[q_j^{(P)} + \left(1 - q_j^{(P)}\right) \prod_{i \in [1,p] | x_{ij} > 0} q_i^{(V)} \right] \end{aligned} \tag{5}$$

Since, by construction, any feasible solution includes at least one PM with at least one VM assigned to it, both product terms in (5) are over non-empty sets.

D. Problem Formulation

The aim of the system operator is to derive an optimal assignment X^* which minimizes cost while maximizing availability. These two criteria are conflicting: the more the VMs deployed and the PMs used to host them, the less the risk of service unavailability, but, at the same time, the higher the cost of the deployment. Since it is not possible to optimize for both criteria simultaneously, we apply a scalarization approach to transform the problem to a single-objective one. The relative importance of the two criteria in deriving an optimal assignment is dictated by a specific *policy*, which is encoded in a pair of weights w_a and w_c (resp. availability and cost) such that $w_a + w_c = 1$. Given a specific policy, the system operator derives the optimal solution to the following problem:

$$\underset{X}{\text{minimize}} \quad w_c\, C(X) - w_a\, A(X) \tag{6}$$

$$\text{subject to} \quad C(X) \leq E \tag{7}$$

$$A(X) \geq A \tag{8}$$

$$\sum_{j=1}^{m} (x_{ij} > 0) \leq 1, \forall i \in [1, p] \tag{9}$$

$$\sum_{i=1}^{p} \sum_{j=1}^{m} x_{ij} = p \tag{10}$$

$$\sum_{i=1}^{p} x_{ij} \leq C_j, \forall j \in [1, m]. \tag{11}$$

To deal with the potential difference in the magnitude of the two components of the objective function, the values of $C(X)$ and $A(X)$ are appropriately normalized in the (0, 1) interval using the upper-lower-bound approach [29]. This model supports specific maximum cost and minimum availability constraints (C and A, respectively; see (7) and (8)). Constraint (9) ensures that, for any VM, its vCPU resources are allocated on a single PM, since a VM cannot be split. Note that an assignment can result in this sum being zero for multiple values of *i*. This occurs when the number of VMs to deploy is less than the number of vCPUs to assign, a case typical in practice. Constraint (10) ensures that all vCPUs are allocated, and constraint (11) guarantees that the CPU capacity of each PM is not exceeded.

Resource allocation and virtual function placement problems such as the one that we address in this work are typically shown to be hard to solve by reduction from known packing or knapsack problems [13]. Our case has the particular properties that (i) the function to minimize involves multiple objectives, (ii) it is non-linear and non-separable due to the availability component that it includes and its weighted combination with the cost objective, and (iii) involves a non-linear constraint (8). Such characteristics are known to make such problems harder to tackle [30]. For example, the non-separable integer knapsack problem with the additional restriction that the objective function has quadratic form is already NP-hard [31], [32]. Moreover, we should note that our availability model is generic in the sense that it allows for heterogeneous PMs with different failure probabilities. Even for the simpler version that we consider in the rest of this article, where PMs are assumed identical from a reliability perspective, thus having equal failure probabilities, we show experimentally that deriving exact optimal solutions is computationally expensive by solving the problem using the CPLEX optimizer.

Table II summarizes our notation.

TABLE II
SUMMARY OF NOTATION

p	number of vCPUs to allocate
m	number of available PMs
v	number of VMs
$C(X)$	cost function
$A(X)$	availability function
C_j	capacity of a physical machine j
w_a	availability weight
w_c	cost weight
e_V	fixed cost per VM instantiated
e_P	fixed cost per PM utilized
$q_i^{(V)}$	failure probability of VM i
$q_j^{(P)}$	failure probability of PM j

E. Relevance With ETSI NFV

Our model and algorithms reflect the particular characteristics of the vCDN use case described by ETSI [2]. From a service viewpoint, the ETSI vCDN use case specifically targets video distribution; our overall design is centered around video delivery, too. The scenario described in the aforementioned specification involves a centralized CDN controller and a number of cache VNF instances distributed across data centers. These caches are identical from a functional perspective. This is in line with our model, where we are interested in allocating CPU resources to identical VNF instances following specific criteria, constraints and performance targets. We are focusing on the placement of cache VNF instances, as these are the ones which will mainly absorb the load of the vCDN, not considering in our algorithms the placement of other components of a vCDN instance. Caches are relatively simple network functions and, as such, it is reasonable to assume that there is a 1:1 mapping between the cache instance and the VM that hosts it, treating it as a VNF composed of a single VNF component. This is the reason why in this article we use the terms VM and VNF instance interchangeably.

Furthermore, at the heart of our model is the cost-availability tradeoff, with both criteria appearing in the objective function and the constraints. This makes our model particularly relevant to NFV. The ETSI has published specifications on NFV resiliency requirements [6], explicitly mentioning the issues and tradeoffs we face and try to capture with our model. The relevance of our work with NFV from a modeling perspective is further established by the fact that, as per the ETSI specifications, there is a requirement for service availability to be defined on a service basis and to be considered together with the total cost and its importance for the customers (in our case content providers). The policy mechanism that we have introduced serves exactly to this end. Different policies and availability constraints can be thus put in effect for different service instances.

As a final note, although the focus of this work is on CDN, other types of services are not out of the question. We remark that our model and algorithms could be extended with minor adaptations towards other families of virtualized applications, provided that a similar cost and availability model are assumed, and that the VNF instances to place have identical functionality, as is the case for CDN caches. Such potential extensions are a topic for further study.

V. SOLVING A RELAXED VERSION OF THE PROBLEM

Finding an exact optimal solution to the problem can be prohibitive computationally. We thus propose an efficient heuristic algorithm to derive solutions which we show to be near-optimal in Section VII. In particular, instead of jointly deciding on optimally distributing the number of vCPUs to VMs and placing the latter in PMs, we tackle the two subproblems separately in two stages: First, we decide on the appropriate number of VMs to utilize and, second, on their actual placement on PMs and the CPU resource distribution to them. Both steps of our *Two-step Resource Allocation and Placement (TRAP)* heuristic algorithm can be efficiently solved in polynomial time. Therefore, the overall procedure of deriving an appropriate resource allocation and placement following a customer's CDNaaS instance deployment request involves three sub-problems outlined below:

1) Find an optimal number of vCPUs to allocate per region to serve customer demand. We solve this problem using the procedure proposed in our prior work [7].
2) Decide on the optimal number of VMs to launch under a specific policy, satisfying cost and availability constraints.
3) Place those VMs on an optimal number of physical hosts and distribute vCPUs among them using a suitable placement algorithm, having as input the results of the previous steps (optimal number of VMs, number of vCPUs). Specific host capacity, cost, and availability constraints apply.

Sections V-A and V-B detail our methods to solve the second and third subproblems respectively. We address each subproblem independently, providing for each one a problem formulation and an algorithm to optimally solve it.

We should note that, for each subproblem, the same minimum availability constraint value (A) as in the original problem is applied, although the availability functions themselves may differ. For the management cost, there is a specific budget for each subproblem (E_V and E_P for the VM- and the PM-level problems respectively), such that $E_V + E_P = E$. The system operator is free to decide how the overall budget is split and various methods for this may be possible. For example, a split ratio $s \in (0,1)$, such that $E_V = sE$ and $E_P = (1-s)E$, can be selected by defining a parameter $\epsilon \in (0,1)$ and successively solving the two subproblems for each $s \in [\epsilon, 2\epsilon, \ldots, 1)$, eventually selecting the assignment which minimizes (6). This procedure involves $\lfloor 1/\epsilon \rfloor$ executions of the proposed algorithms for the two subproblems. If this method is applied, the operator can tune ϵ to force a specific number of iterations.

A. Deciding on the Number of VMs to Launch

In this section, we describe our solution to the second subproblem, which consists in finding the optimal number of VMs to launch. We define an objective function to minimize, which includes a management cost and an availability component.

1) Availability: Given a known VM failure probability q_V, assumed to be fixed and identical for all VMs of the same type included in a vCDN instance (VMs of a specific type, e.g.,

streaming servers, are considered identical and their failure probability does not depend on the resources allocated to them nor their workload), we define service availability at the VM level as

$$A_V(x) = 1 - q_V^x, \tag{12}$$

where x is the number of VMs to deploy. This expression for availability corresponds to the probability that at least one VM is available and is heuristic in the sense that it ignores PM failures and considers VM failures independent.

2) Management Cost: The management cost at the VM level is given by (1) and is assumed linear on the number of VMs used. A simplified expression defined in terms of the number x of VMs to deploy is

$$C_V(x) = e_V x. \tag{13}$$

3) Problem Formulation: Applying the specified policy expressed as the combination of weights $(w_c,\ w_a)$, the function to minimize at the VM level becomes

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & w_c C_V(x) - w_a A_V(x) && (14)\\ \text{subject to} \quad & A_V(x) \geq A, && (15)\\ & C_V(x) \leq E_V, && (16)\\ & x \geq m_{min}, && (17)\\ & x \leq p, && (18) \end{aligned}$$

where A, E_V are the respective constraints of availability and cost, and m_{min} the minimum number of PMs capable of accommodating the required number of vCPUs. The latter is given by

$$m_{min} = \min\left\{ l \in [1, m] | \sum_{i=1}^{l} C_i \geq p \right\}, \tag{19}$$

where $C_1 \geq C_2 \geq C_3 \geq \cdots \geq C_m$ are the capacities of the m PMs of a given region in non-increasing order.

The objective (14) is to find an optimal number of x under availability (15) and cost (16) constraints. Constraint (17) ensures that x is such that no single VM is assigned more vCPUs than the maximum single-host available capacity. If x is not large enough, and given that the number of vCPUs to distribute to these VMs is fixed, and at most x PMs will eventually be used (we cannot use more PMs that the VMs to deploy), it could happen that there exists no subset of x PMs with enough aggregate vCPU capacity (i.e., at least p). This would mean that at least one VM would need more vCPUs than even the highest-capacity PM could offer. For instance, when $x = 1$ (with p vCPUs to distribute), if there is no physical host with a capacity superior to p, such a solution is infeasible. On the other hand, (18) ensures that the VMs to be launched will not be more than the number of vCPUs to allocate.

4) Selection of the Optimal Number of VMs: In order to find the optimal number of VMs by solving (14) under a specific policy, we propose an algorithm whose inputs are (i) the total number, p, of vCPUs to be allocated across the region's hosts in order to serve the customer request, derived in the first phase, (ii) the constraints per objective (A, E_V), and (iii) the combination of weights (w_c, w_a) that defines the adopted policy.

We first observe that the objective function is convex and the value that minimizes its continuous version with $x \in \mathbb{R}_{>0}$ is $x_0 = \log_{q_V} \frac{w_c e_V}{w_a \ln q_V^{-1}}$. Constraints (15) and (16) can be rewritten as $x \geq \log_{q_V}(1 - A)$ and $x \leq \frac{E_V}{e_V}$, respectively. Constraints (15) and (17) thus provide a lower bound to the optimal value of x. Which of the two constraints is more restrictive depends on the configuration of the problem. Similarly, an upper bound is provided by the cost constraint (16), and (18).

To minimize (14) we calculate the value $x_0 \in \mathbb{R}_{>0}$ which minimizes the continuous version of (14). If x_0 lies within the feasible region defined by the above bounds, we select the closest (feasible) integer value to x_0 as the optimal. Otherwise, we select the value of the constraint that x_0 violates.

We should note that in other problem settings where the objective function is not convex, an optimal solution can be found in pseudo-polynomial time in p by evaluating the objective function and the constraints for each $x \in [m_{min}, p]$. The case $x = m_{min}$ corresponds to a deployment which minimizes the number of deployed VMs, while $x = p$ to a deployment that performs a one-to-one vCPU-to-VM assignment. This algorithm runs efficiently for realistic values of p.

B. VM Placement and vCPU Distribution

With the number of VMs v to launch in place, we need to decide on their suitable placement on the underlying PMs and the distribution of the vCPUs among them. The output of this step is an assignment $X = (x_{ij})$, with $i \in [1, v]$ and $j \in [1, m]$, which represents a heuristic solution to (6).

Similar to how we treated the sub-problem of selecting an optimal number of VMs, we define appropriate availability and cost functions at the PM level as follows.

1) Availability: To measure a solution's availability, we use (5), assuming identical PMs with a known failure probability fixed to $q_P^{(j)} = q_P, \forall j \in [1, m]$.

2) Management Cost: Assuming identical PMs, and in turn a fixed management overhead for each PM on which we are placing the customer's VMs, irrespective of their number and the service workload imposed, we model PM-level cost as a linear function of the number of PMs eventually used for hosting at least one VM. This cost is given by (2), substituting p for v in the upper limit of the second summation, since the number of VMs in this case is already known, as calculated at the second step of our scheme.

3) Problem Formulation: We propose the following multi-objective formulation for the problem of placing v VMs to a subset of the available m PMs and distributing p vCPUs to them under capacity, availability and cost constraints, and given policy (w_c, w_a):

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & w_c C_P(X) - w_a A(X) && (20)\\ \text{subject to} \quad & A(X) \geq A, && (21)\\ & C_P(X) \leq E_P, && (22) \end{aligned}$$

$$\sum_{j=1}^{m} (x_{ij} > 0) = 1, \forall i \in [1, v] \quad (23)$$

$$\sum_{i=1}^{v} x_{ij} \leq C_j, \forall j \in [1, m] \quad (24)$$

where A, E_P are the respective availability and cost constraints.

Constraint (23) ensures that a VM is allocated to only one PM, and the set of constraints (24) guarantees that the available capacity per PM is not exceeded.

4) An Algorithm for VM Placement and vCPU Distribution: We propose the following algorithm to derive an optimal solution to (20) in polynomial time, whose input is (i) the number, *p*, of vCPUs to be allocated, (ii) the policy and constraints per objective, (iii) the sorted PMs in non-increasing order of capacities, i.e., $C_1 \geq C_2 \geq C_3 \geq \ldots \geq C_m$, (iv) m_{min}, and (v) the number, *v*, of VMs; the latter three are obtained in the previous step.

This algorithm consists in creating and evaluating a candidate assignment of VMs to m' PMs, for each $m' \in [m_{min}, \min(m, v)]$. More than *v* PMs are not necessary, since that would directly mean that at least one PM would not host any VM. We observe that the value of the objective function only depends on the number of PMs used and the number of VMs assigned to each PM. Our algorithm starts by carrying out the following steps for each m':

1) Distribute the *v* VMs among the m' first PMs proportionally to their capacities.
2) Evaluate the objective function and the constraints for the derived assignment.

Our algorithm returns the assignment *X* of VMs to PMs which minimizes the objective function for the given policy, or responds that there is no feasible solution under the given constraints.

This procedure has $\mathcal{O}(m \times \min(m, v))$ complexity: The two steps have to be repeated at most min(*m,v*) times. Step 1 takes time linear on the number of PMs. (Using the *largest remainder* apportionment method for proportional distribution is $\mathcal{O}(m)$.) Evaluating the availability function could also be implemented to take $\mathcal{O}(m)$ time, if we separately keep track of the number of VMs assigned per PM in a candidate solution.

If a feasible solution using m^* PMs is found, the algorithm ends by:

1) Distributing the *p* vCPUs among the first m^* PMs proportionally to their capacities.
2) For each of the first m^* PMs, distributing the number of the allocated vCPUs evenly to the VMs assigned to it.

The complexity of the algorithm is dominated by the first two steps. Finally, since only the number of PMs and VMs, and the placement of the latter influence the objective function value, any method of sharing vCPUs to PMs/VMs is equivalent.

C. A Note on Network Resource Allocation

Our model and algorithms are oriented towards computing resources; allocating network resources is not in its scope. A basic underlying assumption of this work is that for a given level of user demand for video traffic (which has been translated to specific compute resources), it is more straightforward for the operator of the CDNaaS infrastructure to calculate the amount of necessary network resources. Then, it can accordingly provision the network paths between the virtual instances and end users, and/or perform the necessary traffic engineering tasks.

We should also remark that with appropriate load balancing policies in place (this is straightforward to implement at the CDN service level), a VM with more vCPUs assigned would receive more video requests. Therefore, the traffic that a VM (and in turn a PM) receives is proportional to the amount of compute resources allocated to it. This means that processing resources can be directly linked/translated to networking ones. The system operator then has to provide as input to the algorithm the number of CPU resources per host for which it can guarantee the availability of network capacity necessary for the traffic these vCPUs will handle, which as we argue here is realistic to assume.

Therefore, in this scenario where CPU and network capacities are tightly coupled, our solutions indirectly capture network constraints. However, this would not necessarily hold true if network parameters were part of the objective function to optimize. For example, if network link reliabilities need to be considered in the service availability model or network link costs need to be accounted for, our model and algorithms would have to be reconsidered.

VI. A Genetic Algorithm As an Alternative Approach

A question that may naturally emerge is whether applying *meta-heuristics*, such as Genetic Algorithms (GA), could offer high-quality solutions with reasonable computational cost. A GA generally operates as follows. Each potential solution to the problem is encoded as a string (chromosome) of properties (genes) and is characterized by a fitness value, which is an expression of the solution's quality. Starting from an initial population of candidate solutions (chromosomes), a GA iteratively applies genetic operations on them to produce new generations of better quality. Genetic operations include crossover, where new chromosomes are generated by selected parents, and mutation, where a single chromosome is randomly altered.

In this direction, we build on the approach of Xu and Fortes [9] who, as in our case, deal with a related multi-objective VM placement problem. In particular, they propose a Grouping Genetic Algorithm (GGA) to overcome the limitations of standard GAs when dealing with grouping problems and solve the problem of instantiating VMs on a set of physical hosts aiming to simultaneously minimize resource wasteage, power consumption, and maximum thermal dissipation. These objectives are conflicting, since consolidating VMs in a small number of PMs may lead thermal dissipation to increase significantly in loaded PMs, albeit minimizing the first two objectives.

Inspired by the GGA of Xu and Fortes and adapting it to our problem settings, we have appropriately modified it to integrate our cost and availability objectives. Furthermore, Xu and Fortes apply a fuzzy logic system as a fitness function to jointly consider the conflicting objectives. In our case, however, the preferences with respect to the objectives are known a priori and are expressed using the combination of weights (policy). We thus apply directly our weighted objective function of (6) as the fitness function. Moreover, their algorithm takes as input the number of VMs to place. To obtain this input, we execute the first phase of our heuristic which determines the number of VMs to deploy and then use the GA to select their placement on PMs. After having decided on the number of VMs (see Section V-A), our adapted version of the GA starts by creating an initial population of chromosomes by generating *S* random assignments of VMs to PMs. For each solution in *S*, each group of VMs of a PM represents a gene. Then, the algorithm operates iteratively for *G* generations.

In each generation, as Xu and Fortes suggest, the *ranking-crossover* operator is applied, which is an improved version of the classical crossover. In particular, the advantage of ranking-crossover lies in the fact that it does not select genes blindly from the selected parents, but instead attempts to pick the ones which are ranked higher according to the average value of specific *efficiency* functions defined per objective, thus expected to produce higher quality offspring.

In our context, for minimizing cost, a good solution uses a minimum of virtual and physical resources, while maximizing availability requires instantiating more VMs and spreading them across more PMs. For each ranking-crossover operation, the algorithm selects two random chromosomes as parents and calculates the efficiency values for each involved gene (i.e., group of VMs assigned to a PM). In our algorithm, the efficiency function is defined as the euclidean distance between the pair of values of availability and cost for a gene, given respectively by (12) and (13) with *x* representing the number of VMs contained by the gene, and a *centroid*. We define the centroid as a "utopian" point which represents the pair of cost-availability values considered ideal (and probably not attainable by a single solution at the same time; e.g., a solution with minimum cost and maximum availability). The efficiency function (*object-centroid distance*) of a gene is thus given by

$$\mathcal{E}(c, a) = \sqrt{(u_c - c)^2 + (u_a - a)^2}$$

where the centroid (u_c, u_a) is the vector of ideal values of the cost (u_c) and availability (u_a) objectives and (c, a) represents the obtained cost and availability values for the gene.

Our algorithm ranks the genes of the parents in increasing order of the object-centroid distance and creates a new chromosome by combining the highest-ranking genes, i.e., the ones with the smallest values of $\mathcal{E}$.

For each generation round, this procedure is repeated according to the desired crossover rate, r_c, which determines the number of new chromosomes to be generated per round. Typically, the crossover rate is set to a value between 0.5 and 1, producing $r_c S$ offspring on average. The eventual solution pool population at the end of a generation is constructed by evaluating the fitness function for each chromosome and keeping the top-*S* of them. The algorithm is terminated after *G* generations, where the solution with the minimum value of (6) is returned. According to the observations of Xu and Fortes, mutation does not offer tangible advantages in their case. We therefore chose not to perform mutation either.

Xu and Fortes report that the complexity of their Grouping Genetic Algorithm is $\mathcal{O}(SNlogN + NSG)$, where *N* is the number of VMs to place. The first term is due to the execution of the First-Fit placement algorithm to generate the initial pool of *S* solutions and the second for generating and evaluating all potential placements for *G* generations.

Our GGA has a slightly different complexity. Instead of performing First-Fit, we generate the initial pool of *S* solutions by randomly placing *N* VMs over the *m* PMs in $\mathcal{O}(SN)$ time. Then, each of the *G* generation rounds involves $r_c S$ ranking-crossover operations and the selection of the population of the next generation. For each of the two parents involved in a crossover, calculating the efficiency of up to *m* genes (in case all *m* PMs are utilized) requires $\mathcal{O}(m)$ time, sorting the genes of the parents according to their efficiency is $\mathcal{O}(mlogm)$, and creating a new offspring is $\mathcal{O}(m)$. This is repeated $r_c S$ times yielding an $\mathcal{O}(Smlogm)$ time per generation. Finally, selecting the top-*S* solutions of the population which also includes the offspring requires evaluating the fitness function, i.e., computing the objective function of (6), for each solution in the population. This be carried out in $\mathcal{O}(m)$ time (see Section V-B4) per chromosome, while selecting an (unordered) set of the top-*S* chromosomes of the population to create the next generation is $\mathcal{O}(S(1 + r_c)) = \mathcal{O}(S)$. The computation time for each generation is thus dominated by the creation of offspring and the overall complexity of our GA is $\mathcal{O}(SN + GSmlogm)$.

VII. Performance Evaluation

We evaluate the performance of our algorithms under two perspectives. First, we aim to demonstrate how our model and algorithms address the cost-availability tradeoff and show how the selection of specific policies drives the solutions towards different optima. Second, we compare the performance of our scheme (TRAP) with alternative approaches in terms of the quality of the produced solutions (i.e., how close they are to the optimal) and the execution time. Note that for the latter we implemented our model in the CPLEX[3] environment and used its optimizer to derive exact optimal solutions as a benchmark.

A. Cost vs. Availability Tradeoff

We begin with an experiment where we run our algorithms under different combinations of weights (w_c, w_a), each corresponding to a different policy and for the same problem settings. In particular, we use (i) the same number of vCPUs to allocate, (ii) the same number of available hosts and their capacities, and (ii) the same cost and availability constraints. We set the availability constraint to "five nines" (99,999%), a popular availability target for carrier-grade NFV. We fix the

[3]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

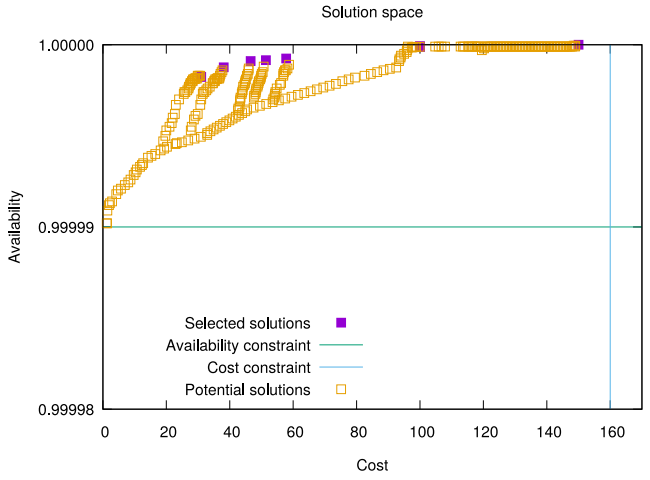

Fig. 2. Solution space. The straight lines represent cost and availability constraints. The filled boxes are the assignments selected as optimal by our TRAP algorithm under different policies, while the empty boxes are feasible but suboptimal solutions.

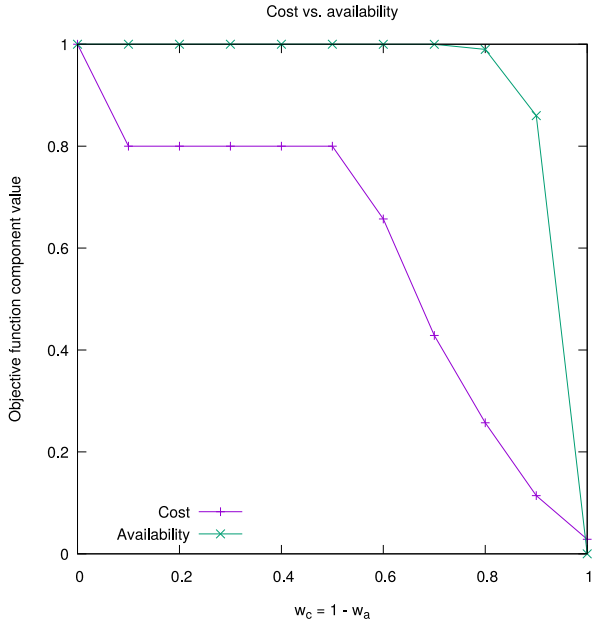

Fig. 3. Availability and cost as a function of the selected policy.

overall cost constraint to $E = 160$ and assume that the costs for a single VM and PM are $e_V = 1$ and $e_P = 1$ respectively. The failure probabilities for VMs and PMs are set to $q_V = q_P = 0.001$. Unless otherwise noted, these values are used all across the experiments of this section. We consider a system with 50 PMs, each with a capacity between 2 and 15 vCPUs selected uniformly at random.

Fig. 2 presents the solution space, where the x-axis and y-axis represent the absolute values of the cost and availability functions respectively. Each filled square point corresponds to the outcome of the TRAP algorithm (a vCPU-VM-PM assignment identified as the optimal) according to a specific policy, while empty squares are feasible but suboptimal solutions. The vertical and horizontal lines are the respective cost and availability constraints and limit the space of feasible solutions. Note that in our tests it is possible that for two different policies the same optimal solution is derived. These points are superimposed in the figure.

The selected solutions represent what our algorithm identifies as the *Pareto frontier*, which, in our context, is the set of solutions for which it is not possible to improve on cost without sacrificing on availability and vice versa. Each such solution represents a different availability-cost trade-off. A cost-centric policy, e.g., $(w_c, w_a) = (0.8, 0.2)$, guides our algorithm to select as the optimal a low-cost solution, which, according to the cost function, uses a small number of VMs/PMs. In turn, this corresponds to a lower service availability. (The algorithm guarantees that it is a feasible solution, not violating the 0.99999 availability constraint.) Availability is increasingly higher (≈ 1) when more hosts are used. Such solutions correspond to increasing w_a values, which our mechanism takes into account to drive the calculation towards an appropriate VNF placement which uses more virtual instances and PMs.

This is more evident in Fig. 3, where we detail the evolution of the value of each objective (cost and availability) in each optimal solution as a function of the given policy. For reasons of clarity, the values presented are normalized by mapping the lowest and highest value per objective to 0 and 1 respectively. The figure indicates that as the weight of the cost objective w_c increases, both functions are decreasing. This is positive from a cost but not from an availability perspective, and is due to less VMs/PMs being used as w_c approaches 1 and w_a approaches 0. (The inverse holds for availability.)

B. Performance Comparison

We compare TRAP with the following placement algorithms:

- *Random:* Given a number of VMs, *Random* places each VM at an available host selected uniformly at random.
- *First-Fit (FF):* This algorithm places each VM at the first available machine that has the capacity to host it.
- *Genetic Algorithm (GA):* This is the grouping genetic algorithm we proposed in Section VI.
- *CPLEX:* This is an exact algorithm which derives the optimal solution to (6) using the CPLEX optimizer.

Random, *FF* and *GA* require the number of VMs to place as input. In the experiments presented in this section, it is implied that this input is provided by executing the first step of TRAP. This improves the performance of these schemes compared to a random or policy-unaware decision for the number of VMs to place. As our results show, though, our two-step heuristic still brings about significant performance gains.

In our comparison, our metric for the quality of a solution is the ratio of the objective function value of the solution returned by the algorithm in question to the respective value of the (exact) optimal solution returned by CPLEX. This is expressed as a percentage and is denoted as *optimality*. *CPLEX* corresponds to an optimality of 100%.

Fig. 4 presents a comparison of TRAP (purple curve, cross points) with our grouping genetic algorithm (green curve, "x"

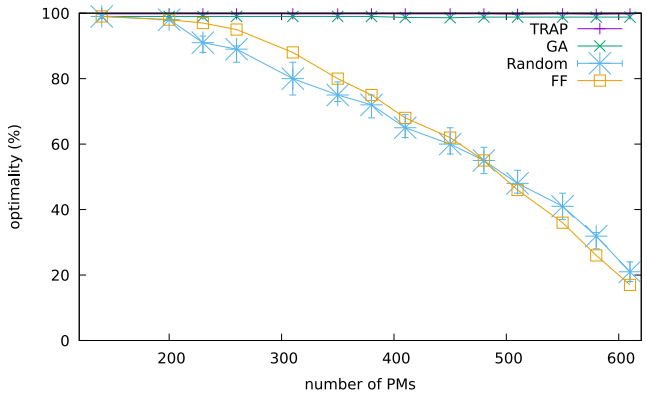

Fig. 4. Performance of different placement schemes for the same policy and increasing numbers of available PMs.

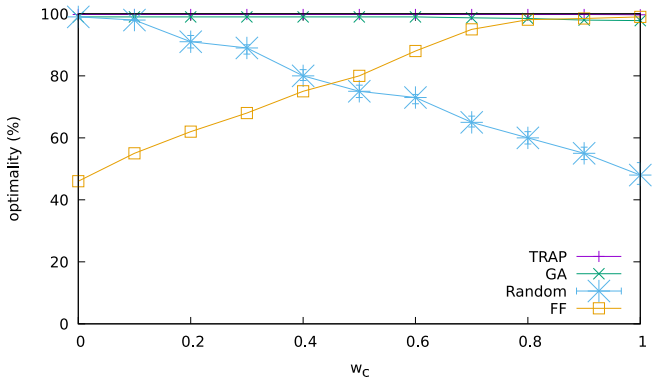

Fig. 5. Performance of different placement schemes for the same configuration (number of available PMs) as a function of the applied policy (criteria weights).

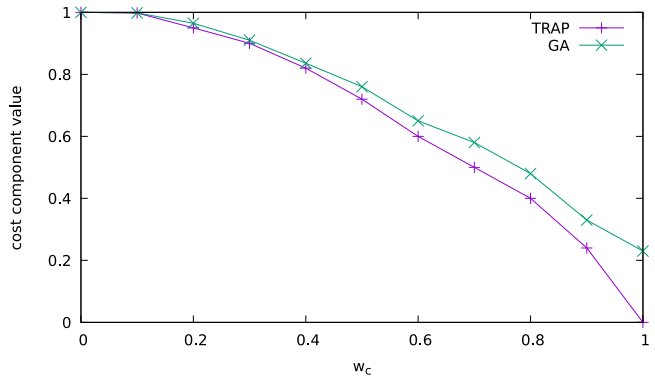

Fig. 6. Comparison between our two-step heuristic (TRAP) and our GA in terms of the cost objective for different policies (lower is better).

points), random (blue curve, "*" points), and first-fit placement (yellow curve, square points) in terms of optimality as a function of the number of available PMs.

To obtain these results we ran the five schemes under the same configuration, which corresponds to the same policy (in this case, $w_c = w_a = 0.5$), the same number of PMs, each with available capacity selected uniformly at random and ranging from 2 to 15 vCPUs, $p = 700$ vCPUs to distribute, and the same capacity constraint. Each point is the mean of 50 iterations for the same configuration except for PM capacities which vary, presented with 95% confidence intervals.

As can be seen from Fig. 4, TRAP approximates well the optimal solution. In particular our heuristic gives objective function values that are very close to the ones returned by CPLEX. GA is also remarkably close to the optimal solution, but is consistently outperformed by our heuristic and at the same time comes with significant processing overhead, as we shall show.

Random and FF placement, on the other hand, result in solutions that are increasingly far from the optimal as the number of available PMs increases, which is due to the fact that they do not take into account the main objectives and the respective policy. With an increase in the number of available PMs, the solution space also expands and these algorithms fail to "explore" it effectively. *Random* tends to result in solutions which utilize increasing numbers of PMs thus driving cost up, while *FF* tends to consolidate VMs in the least number of PMs, which reduces availability.

Then, we carry out an experiment where we fix the number of available PMs and measure how each of the candidate algorithms approaches the optimal solution for different policies. Fig. 5 confirms that our heuristic tops the candidate schemes in terms of optimality. Importantly, policy-unaware schemes cannot perform consistently across the spectrum of available policies. For configurations with many available PMs, such as the one in this experiment, and when only availability is critical ($w_c \to 0$), *Random* may have acceptable performance since it will tend to distribute VMs uniformly across PMs. This advantage quickly diminishes as we move towards more cost-centered policies. Conversely, *FF* performs near-optimally when availability is not a concern and cost minimization is what mainly matters ($w_c > 0.7$), since it aims to pack VMs in as few PMs as possible. On the other hand, our two-step heuristic and our genetic algorithm, which incorporate policies in their design, address successfully the cost-availability tradeoff, producing solutions near the optimal for all weight combinations.

If we delve further into the performance of these two algorithms and consider the two objectives separately, we notice that TRAP achieves better performance in terms of both cost and availability at the same time, compared with the genetic algorithm. Fig. 6 presents the (normalized) value of the cost component for different policies. As optimizing for cost becomes more important, the performance improvement of TRAP compared to the GA grows. The solutions derived using this scheme also improve on availability, especially when the applied policy seeks for a balance between the two objectives, as Fig. 7 indicates. The gains in terms of optimality that our heuristic brings about compared to our GA can be attributed, to an extent, to the fact that the latter creates the original pool of solutions by randomly assigning VMs to PMs. In some cases, apart from slower convergence, this may affect the quality of the solutions produced by the GA.

Although both our heuristic and our GA approximate well the optimal solution, they come with different processing overheads. To quantify their running time performance, we carried out an experiment where we fixed the policy to $w_c = w_a = 0.5$

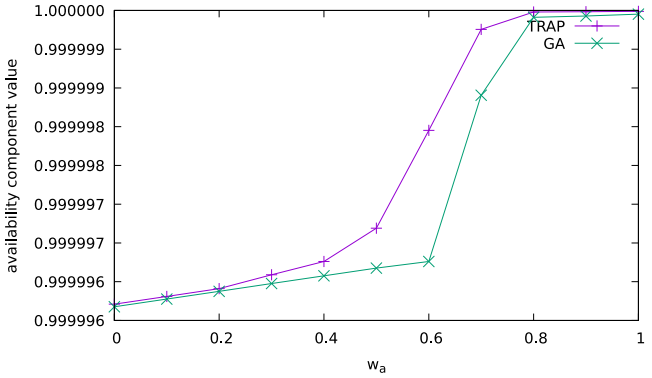

Fig. 7. Comparison between our two-step heuristic (TRAP) and our GA in terms of the availability objective for different policies (higher is better).

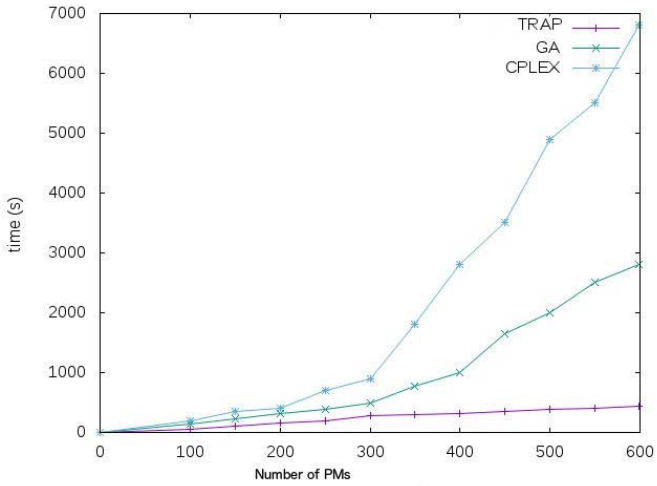

Fig. 8. Execution time as a function of the number of available PMs for different algorithms.

and measured the execution time of the CPLEX solver, the TRAP algorithm, and the GA. Regarding the GA, and as in our previous experiments, we configure the size of the solution pool to $S = 70$ and the number of generations to $G = 20$. These numbers were selected experimentally and represent a good compromise between execution time and the quality of the derived solutions. For reasons of a fair comparison, the execution time reported for the GA also includes the time it takes to determine the number of VMs to launch, for which we apply the first step of our heuristic; the GA then proceeds by selecting their appropriate placement. Our experiments were performed on an Intel i7 machine with 8 CPU cores and 8 GB of RAM, running Ubuntu 14.04.

Fig. 8 verifies that our TRAP scales well as the number of PMs grows. Even for 600 PMs, it produces a solution which is close to the optimal in the order of few minutes. The GA scales poorly for such problem instances, which is justified by its higher computational complexity (see Section VI). Its running time performance could be improved by reducing the number of generations or the solution pool size, but this would come at the expense of the quality of solutions it produces, which are already suboptimal to the ones of TRAP. Note that for more than 600 PMs, it was not tractable to derive the exact optimal solution using CPLEX, since its execution time increases dramatically when the number of variables and/or constraints of the model increases.

VIII. Conclusion

This article presented our study on the problem of jointly allocating compute resources to virtual instances and their placement on an NFVI for the provision of CDN-as-a-Service. Our focus was on simultaneously addressing the conflicting requirements for improved service availability and reduced management cost. To this end, we proposed a multi-objective optimization formulation of the problem, as well as efficient heuristic algorithms to solve it. We demonstrated quantitatively how our algorithms optimally address the cost-availability tradeoff. By comparing our scheme to simple baseline algorithms that are often used as benchmarks in related work, we demonstrated that in order to address more effectively both objectives, a flexible scheme which incorporates their relative importance in its design is necessary. Our approach can prove beneficial for the operator of such a system in order to appropriately dimension a virtualized CDN service, implementing resource allocation policies that reflect customer (i.e., content provider) preferences, and assisting in the establishment and enforcement of specific service-level agreements. The heuristic algorithms we devised run efficiently, without sacrificing on solution quality. We have shown experimentally that they can derive near-optimal solutions in the order of few minutes for large problem instances, where exact solutions by means of solvers such as CPLEX are intractable to get. Finally, we explore alternative strategies to tackle our problem by proposing a genetic algorithm suitable for our model, inspired by the related state-of-the-art. Our heuristics were shown to outperform the genetic algorithm, both in terms of optimality and efficiency.

Although the proposed scheme allows deriving a near optimal solution to place vCDN resources over a telco or federated cloud, it requires a special focus when fixing the weight values, which mainly rely on the vCDN service type to deploy. For example, for a vCDN that delivers highly-popular content, higher reliability may be required by the content provider, and thus w_a should be significantly higher than w_c. For deployments which deliver less popular content or which involve short service duration, on the other hand, the customer might be less interested in high availability guarantees. In these cases, the operator might select a cost-centered policy with a high value for w_c to reduce operational cost and possibly to be able to provide a more affordable offer to the customer. Therefore, the service type has critical impact on the selected deployment policy.

Our work on the subject is ongoing and focuses on cases which our model and algorithms cannot currently address. One direction is adapting our cost functions to consider whether cloud hosts are already active hosting virtual instances; it could be argued that favoring such PMs in the placement process would lead to more significant energy savings. Another line of research studies extensions of our design towards edge computing. The main challenges therein lay in the inherent scarcity of edge resources, and the heterogeneity naturally

introduced in the underlying physical infrastructure both in terms of operating cost and reliability. On the other hand, edge computing offers the potential for lower latency and thus improved user experience while saving on the operator's backhaul network capacity. These new features require modifications both at our model and our algorithms. In view of the upcoming 5$^{\text{th}}$ Generation (5G) mobile networks, these modifications will need to account for more dynamic environments with the appropriate architectural support [33], where VNF components may need to be shifted across (edge and other) hosts following shifts in user demand or the conditions in the NFVI. Finally, extensions to account for failure recovery costs are a topic for future study.

REFERENCES

[1] *Network Functions Virtualisation (NFV); Management and Orchestration*, ETSI Standard GS NFV-MAN 001, Dec. 2014.

[2] *Network Functions Virtualisation (NFV); Use Cases*, ETSI Standard GS NFV 001, Oct. 2013.

[3] B. Frank *et al.*, "Collaboration opportunities for content delivery and network infrastructures," *ACM SIGCOMM Ebook Recent Adv. Netw.*, vol. 1, pp. 305–377, Aug. 2013.

[4] N. Herbaut, D. Négru, Y. Chen, P. A. Frangoudis, and A. Ksentini, "Content delivery networks as a virtual network function: A win-win ISP-CDN collaboration," in *Proc. IEEE Globecom*, 2016, pp. 1–6.

[5] P. A. Frangoudis, L. Yala, and A. Ksentini, "CDN-as-a-service provision over a telecom operator's cloud," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 702–716, Sep. 2017.

[6] *Network Functions Virtualisation (NFV); Resiliency Requirements*, ETSI Standard GS NFV-REL 001, Jan. 2015.

[7] L. Yala, P. A. Frangoudis, and A. Ksentini, "QoE-aware computing resource allocation for CDN-as-a-service provision," in *Proc. IEEE Globecom*, 2016, pp. 1–6.

[8] L. Yala, P. A. Frangoudis, G. Lucarelli, and A. Ksentini, "Balancing between cost and availability for CDNaaS resource placement," in *Proc. IEEE Globecom*, 2017, pp. 1–7.

[9] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. (GreenCom)*, 2010, pp. 179–188.

[10] J. Dong *et al.*, "Energy-saving virtual machine placement in cloud data centers," in *Proc. IEEE/ACM CCGrid*, 2013, pp. 618–624.

[11] N. Quang-Hung, N. T. Son, and N. Thoai, "Energy-saving virtual machine scheduling in cloud computing with fixed interval constraints," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXI* (LNCS 10140), A. Hameurlain, J. Küng, R. Wagner, T. Dang, and N. Thoai, Eds. Heidelberg, Germany: Springer, 2017.

[12] D. Gmach, J. Rolia, and L. Cherkasova, "Resource and virtualization costs up in the cloud: Models and design choices," in *Proc. IEEE/IFIP DSN*, 2011, pp. 395–402.

[13] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—A survey of problem models and optimization algorithms," *ACM Comput. Surveys*, vol. 48, no. 1, p. 11, 2015.

[14] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proc. IFIP/IEEE IM*, 2007, pp. 119–128.

[15] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2647–2660, Nov. 2014.

[16] X. Li and C. Qian, "A survey of network function placement," in *Proc. 13th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2016, pp. 948–953.

[17] R. Mijumbi *et al.*, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. 1st IEEE Conf. Netw. Softw. (NetSoft)*, 2015, pp. 1–9.

[18] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[19] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 343–356, Jun. 2017.

[20] M. Mitchell, *An Introduction to Genetic Algorithms*, 5th ed. Cambridge, MA, USA: MIT Press, 1999.

[21] S. Yang, P. Wieder, and R. Yahyapour, "Reliable virtual machine placement in distributed clouds," in *Proc. 8th Int. Workshop Resilient Netw. Design Model. (RNDM)*, 2016, pp. 267–273.

[22] T. Ecarot, D. Zeghlache, and C. Brandily, "Consumer-and-provider-oriented efficient IaaS resource allocation," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2017, pp. 77–85.

[23] M. Casazza, P. Fouilhoux, M. Bouet, and S. Secci, "Securing virtual network function placement with high availability guarantees," in *Proc. IFIP Netw.*, 2017, pp. 1–9.

[24] L. Qu, C. Assi, K. B. Shaban, and M. Khabbaz, "Reliability-aware service provisioning in NFV-enabled enterprise datacenter networks," in *Proc. 12th Int. Conf. Netw. Service Manag. (CNSM)*, 2016, pp. 153–159.

[25] M. Callau-Zori, L. Samoila, A.-C. Orgerie, and G. Pierre, "An experiment-driven energy consumption model for virtual machine management systems," IRISA, Rennes, France, Res. Rep. RR-8844, Jan. 2016.

[26] H. Ouarnoughi, J. Boukhobza, F. Singhoff, and S. Rubini, "A cost model for virtual machine storage in cloud IaaS context," in *Proc. 24th Euromicro Int. Conf. Parallel Distrib. Netw. Based Process. (PDP)*, 2016, pp. 664–671.

[27] A. Israel and D. Raz, "Cost aware fault recovery in clouds," in *Proc. IFIP/IEEE IM*, 2013, pp. 9–17.

[28] *OpenStack—Open Source Cloud Computing Software*. Accessed: Oct. 4, 2018. [Online]. Available: https://www.openstack.org/

[29] R. T. Marler and J. S. Arora, "Function-transformation methods for multi-objective optimization," *Eng. Optim.*, vol. 37, no. 6, pp. 551–570, 2005.

[30] N. Halman, H. Kellerer, and V. A. Strusevich, "Approximation schemes for non-separable non-linear boolean programming problems under nested knapsack constraints," *Eur. J. Oper. Res.*, vol. 270, no. 2, pp. 435–447, 2018.

[31] G. Gallo, P. L. Hammer, and B. Simeone, "Quadratic knapsack problems," in *Combinatorial Optimization* (Mathematical Programming Studies), vol. 12, M. W. Padberg, Ed. Heidelberg, Germany: Springer, 1980.

[32] A. Billionnet and F. Calmels, "Linear programming for the 0–1 quadratic knapsack problem," *Eur. J. Oper. Res.*, vol. 92, no. 2, pp. 310–325, 1996.

[33] G. Bianchi *et al.*, "Superfluidity: A flexible functional architecture for 5G networks," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 9, pp. 1178–1186, 2016.

Louiza Yala received the M.Sc. degree in networking from the University of Bejaia, Algeria, in 2014 and the M.Sc. degree in technologies for information processing and systems analysis from the University of Technology of Compiègne, France, in 2015. She is currently pursuing the Ph.D. degree with the University of Rennes 1, France, and a member of the IRISA/INRIA Team Dionysos. Her research interests include network virtualization, Internet multimedia, content delivery networks, and QoE.

Pantelis A. Frangoudis received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the Department of Informatics, Athens University of Economics and Business, Greece, in 2003, 2005, and 2012. From 2012 to 2017, he was with the Team Dionysos, IRISA/INRIA/University of Rennes 1, Rennes, France, which he joined under an ERCIM Post-Doctoral Fellowship from 2012 to 2013. He is currently a Researcher with the Communication Systems Department, EURECOM, Sophia Antipolis, France. His research interests include mobile and wireless networking, Internet multimedia, network security, future Internet architectures, cloud computing, and QoE monitoring and management.

Giorgio Lucarelli received the Ph.D. degree in computer science from the Department of Informatics, Athens University of Economics and Business, Greece, in 2009. From 2010 to 2017, he held several post-doctoral research positions with different universities in France, specifically with the University Paris-Dauphine, the University Pierre et Marie Curie, and the University of Grenoble-Alpes. Since 2018, he has been an Associate Professor with the University of Lorraine, France. His research interests include the design and analysis of algorithms, scheduling, optimization in graphs, and online algorithms.

Adlen Ksentini received the Ph.D. degree in computer science from the University of Cergy-Pontoise in 2005, with a dissertation on QoS provisioning in IEEE 802.11-based networks. From 2006 to 2016, he was with the University of Rennes 1 as an Assistant Professor and was a member of the Dionysos Team with INRIA, Rennes. Since 2016, he has been an Assistant Professor with the Communication Systems Department, EURECOM. He has been involved in several national and European projects on QoS and QoE support in future wireless, network virtualization, cloud networking, mobile networks, and more recently on network slicing and 5G in the context of H2020 projects 5G!Pagoda and 5GTransformer. He has co-authored over 100 technical journal and international conference papers. He was a recipient of the Best Paper Award from IEEE IWCMC 2016, IEEE ICC 2012, ACM MSWiM 2005, and the 2017 IEEE Comsoc Fred W. Ellersick Prize (Best IEEE Communications Magazine's Paper). He has given several tutorials in IEEE international conferences, IEEE Globecom 2015, IEEEE CCNC 2017, IEEE ICC 2017, IEEE/IFIP IM 2017. He has been acting as the TPC Symposium Chair for IEEE ICC 2016/2017, IEEE GLOBECOM 2017, IEEE Cloudnet 2017, and IEEE 5G Forum 2018. He has been acting as a Guest Editor of the IEEE JOURNAL OF SELECTED AREA ON COMMUNICATION Series on Network Softwarization, the IEEE WIRELESS COMMUNICATIONS, *IEEE Communications Magazine*, and two issues of ComSoc MMTC Letters. He has been on the Technical Program Committees of major IEEE ComSoc conferences, including ICC, GLOBECOM, ICME, WCNC, and PIMRC. He is currently the Director of the IEEE ComSoc EMEA region and the Vice-Chair of the IEEE ComSoc Technical Committee on Software. He is a COMSOC Distinguished Lecturer.