POLITECNICO DI TORINO
Repository ISTITUZIONALE

Dynamic Service Provisioning in the Edge-cloud Continuum with Bounded Resources

Original
Dynamic Service Provisioning in the Edge-cloud Continuum with Bounded Resources / Cohen, I., Chiasserini, C.F., Giaccone, P., Scalosub, G.. - In: IEEE-ACM TRANSACTIONS ON NETWORKING. - ISSN 1063-6692. - STAMPA. - 31:6(2023), pp. 3096-3111. [10.1109/TNET.2023.3271674]

Availability:
This version is available at: 11583/2978031 since: 2023-12-21T07:21:57Z

Publisher:
IEEE

Published
DOI:10.1109/TNET.2023.3271674

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

27 November 2026

Dynamic Service Provisioning in the Edge-cloud Continuum with Provable Guarantees

Itamar Cohen, *Member, IEEE,* Carla Fabiana Chiasserini, *Fellow, IEEE,* Paolo Giaccone, *Senior Member, IEEE,* Gabriel Scalosub, *Member, IEEE*

***Abstract*—We consider a hierarchical edge-cloud architecture in which services are provided to mobile users as chains of virtual network functions. Each service has specific computation requirements and target delay performance, which require placing the corresponding chain properly and allocating a suitable amount of computing resources. Furthermore, chain migration may be necessary to meet the services' target delay, or convenient to keep the service provisioning cost low. We tackle such issues by formalizing the problem of optimal chain placement and resource allocation in the edge-cloud continuum, taking into account migration, bandwidth, and computation costs. Specifically, we first envision an algorithm that, leveraging resource augmentation, addresses the above problem and provides an upper bound to the amount of resources required to find a feasible solution. We use this algorithm as a building block to devise an efficient approach targeting the minimum-cost solution, while minimizing the required resource augmentation. Our results, obtained through trace-driven, large-scale simulations, show that our solution can provide a feasible solution by using half the amount of resources required by state-of-the-art alternatives.**

***Index Terms*—Edge computing, edge-cloud continuum, service placement, programmable networks**

I. Introduction

Today's networks offer an unprecedented level of resource virtualization, available as a continuum from the edge to the cloud [1]–[7]. These virtual resources are embodied as a collection of datacenters that host service function chains. These service chains provide a plethora of applications, including infotainment [2], road safety [3] and virtual network functions [5], [8]–[10]. These applications have versatile service requirements; for instance, a road safety application requires low latency, which may dictate processing it in an edge datacenter, close to the user. On the other hand, infotainment tasks are more computation-intensive but less latency-sensitive, and therefore may be offloaded to the cloud, where computation resources are abundant and cheap [**?**], [4].

Deploying service function chains is even more challenging when dynamic traffic conditions exist and/or some of the users are mobile. In such cases, service chains may need to be migrated in order to follow the mobile user and, thus, reduce latency [1], [2], [11]. However, when the system is highly-loaded, there may not be enough available resources in the migration's destination. Hence, providing reliable service may compel using some over-provisioning, or *resource augmentation* – at the cost of increasing the system's capital expenses. Existing schemes [2], [4], [8], [9], [11]–[15] perform well when the system load is not too high, but fail to provide a feasible solution under a high load of service requests. To the best of our knowledge, no previous work provides guarantees of finding a feasible solution for the problem whenever such a solution exists.

In this work, we study the combined service *Deployment and Migration Problem* (DMP) in a multi-tier network, where the service orchestrator [6] has to decide: (i) where to deploy a service chain across the cloud-edge continuum, (ii) which resources to allocate for each part of every service chain, and (iii) which chains to migrate, and to which datacenter, to fulfill the service requirements while minimizing the overall deployment and migration costs. Our main contributions are as follows:

- We first formalize the DMP, and show that even finding a feasible solution to the problem – regardless of its cost – is NP-hard.
- We take latency as the main Key Performance Indicator (KPI) [16], as specified by the Service Level Agreement (SLA), and show how to calculate the minimal amount of CPU resources required for placing every service chain on any datacenter, while satisfying the latency requirements.
- We develop a placement algorithm that, leveraging some bounded amount of resource augmentation, is guaranteed to provide a feasible solution whenever such a solution exists for the case with no resource augmentation.
- We present an algorithm that, given a feasible solution, greedily decreases its cost, while keeping the required resource augmentation minimal.
- We compare the performance of our proposed solution to those of existing alternatives using two large-scale vehicular scenarios and real-world antenna locations. Our results show that our algorithm can provide a feasible solution using half the computing resources required by existing alternatives. Our evaluation further highlights several system trade-offs, such as the preferred decision period between subsequent runs of the algorithm.

The rest of the paper is organized as follows. After introducing the system model in Sec. II, we formalize the optimal deployment and migration problem in Sec. III, and overview

I. Cohen and G. Scalosub are School of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Beer Sheva, Israel. E-mail: itamar.cohen@polito.it; sgabriel@bgu.ac.il. C. F. Chiasserini and P. Giaccone are with the Department of Electronics and Telecommunications, Politecnico di Torino, Italy, and with CNIT, Parma, Italy. E-mail: first-name.lastname@polito.it

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

our solution concept in Sec. IV. The problem is decomposed into a computational resource allocation problem, studied and solved in Sec. V, and a placement problem, characterized and solved in Sec. VI. Our overall algorithmic solution is described in Sec. VII and its performance is assessed in Sec. VIII. Finally, Sec. IX discusses related work, and Sec. X draws some conclusions.

II. Modeling the edge-cloud architecture

This section introduces the model for the network infrastructure and the services offered to mobile users and describes how we compute the service delay.

A. Network model

We consider a fat-tree edge-cloud hierarchical network architecture. As described in [**?**], the network comprises: (i) *datacenters* (denoting generic computing resources), (ii) *switches* (generic switching nodes, as routers, switches, multiple switches associated with Multi-Chassis Link-Aggregation (MCLA) [17]), and (iii) *radio Points of Access (PoA)*. Datacenters are connected through switches, and PoAs may have a co-located datacenter [18]. Each user is connected to the network through a PoA, which may vary as the user moves. An example of such a system is depicted in Fig. 1.

We denote by $\mathcal{S}$ the set of datacenters, and model the logical multi-tier network as a directed graph $\mathcal{G} = (\mathcal{S}, \mathcal{L})$ where the vertices are the datacenters, while the edges are the directed virtual links connecting them, i.e., $(i, j) \in \mathcal{L}$ with $i, j \in \mathcal{S}$. Let $D(\mathcal{G})$ denote the *diameter* of $\mathcal{G}$, and r the root of the fat tree topology. For any two datacenters i, j, $\mathcal{P}(i, j)$ denotes the directed path from i to j, with $\mathcal{P}(i, j)$ referring to a sequence of physical links, or vertices, depending on the context. We consider that such a path is loop-free and uniquely predetermined between any two vertices.

B. Services and chain deployment

Consider a generic user generating a *service request* u, originating at the PoA p^u to which the user is currently connected. Each service request is addressed through an instance of VNF *chains*, where each VNF is deployed on a dedicated virtual machine (VM) or container in a datacenter. For the convenience of presentation, hereinafter we refer to VMs only. We refer to the instance of the chain for service request u as $\mathbf{h}^u = (v_1^u, \ldots, v_{h_u}^u)$, where h_u indicates the number of VMs in $\mathbf{h}^u$. Let $\mathcal{U}$ denote the set of service requests, and $\mathcal{H}$ the set of corresponding chains that are currently deployed, or need to be deployed, in the network. Furthermore, for every subset of requests $\mathcal{U}' \subseteq \mathcal{U}$, we let $\mathcal{H}' \subseteq \mathcal{H}$ denote the subset of chains corresponding to $\mathcal{U}'$. For simplicity of notation, while referring to VMs and datacenters hosting them, we will drop superscripts and subscripts whenever clear from the context.

To successfully serve chain $\mathbf{h}^u$, the chain should be fully deployed on one of the datacenters on the path from its PoA p^u to root r [3], [5], [19]. We denote that path by $\mathcal{P}(p^u, r)$. Distinct deployment decisions incur distinct *costs* that we detail in Sec. III.

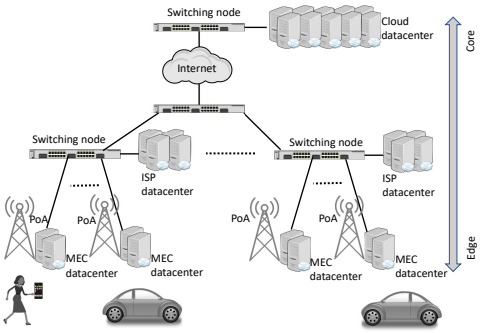


Fig. 1: Example of network scenario for mobile service provisioning.

Each service is associated with an SLA, which specifies the requirements in terms of KPIs [16], and with a maximum amount of resources, e.g., for which the user is willing to pay the network provider. We consider *latency* as the most relevant KPI, although our model could be extended to others, like throughput and energy consumption. We thus associate with each chain $\mathbf{h}^u$ a *target delay* $\Delta(\mathbf{h}^u)$.

C. Service delay

The service delay comprises the computational and the network delays, as detailed below.

Computational delay. Given chain $\mathbf{h}^u$, each VM $v_k^u \in \mathbf{h}^u$ has some input traffic load, λ_k^u, expressed in bit/s, which is known a-priori [4], [11]. In particular, λ_1^u denotes the input traffic to the chain at the PoA associated with the request. We let θ_k^u represent the processing capacity required to handle a single unit of traffic corresponding to v_k^u, expressed in CPU cycles/bit. Thus, $\lambda_k^u \theta_k^u$ represents the CPU cycles/s required to process the incoming traffic. The computation is defined in terms of single data units to be processed. Let γ_k^u be the number of bits per data unit, then $\gamma_k^u \theta_k^u$ is the number of CPU cycles required to process each data unit[1].

Let chain $\mathbf{h}^u$ be placed on datacenter s. For each VM $v_k^u \in \mathbf{h}^u$, $\mu_k^{u,s}$ denotes the processing capacity allocated to such VM on s, expressed in number of CPU cycles/s. As often done in the literature [16], [20]–[22], CPU processing at the VM is modeled through an M/M/1 queue. The average computational delay at VM v_k^u to process one data unit is given by

$$D_k^u(\mu_k^{u,s}) = \gamma_k^u \theta_k^u / (\mu_k^{u,s} - \theta_k^u \lambda_k^u), \tag{1}$$

where we must have $\mu_k^{u,s} > \theta_k^u \lambda_k^u$. The overall computational delay of chain $\mathbf{h}^u$ is thus given by:

$$d^c(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}) = \sum_{k=1}^{h_u} D_k^u(\mu_k^{u,s}). \tag{2}$$

To reflect real-world conditions, when allocating virtual cores to VMs, we consider that $\mu_k^{u,s}$ is an integer multiple of a basic CPU speed, and, thus, $\mu_k^{u,s}$ is discrete and will be expressed in CPU units in the following; θ_k^u is coherently expressed in a fractional value of CPU units.

[1]As an example, a single data unit could be a video frame to process in an object-recognition VNF. Different image resolutions will result in different values of γ_k^u and θ_k^u. In a DPI application, a single data unit would be instead a single data packet.

We denote by $\hat{C}_u$ the maximum amount of computing resources that may be allocated to chain $\mathbf{h}^u$ as per the SLA. Each datacenter $s \in \mathcal{S}$ has a total processing capacity C_s, expressed in number of CPU cycles/s. It is fair to assume that $\hat{C}_u$ does not exceed the processing capacity of any single datacenter in the system, i.e., $\forall s \in \mathcal{S}, u \in \mathcal{U}, \hat{C}_u \leq C_s$.

Network delay. The intra-datacenter communication delay is typically negligible [23] compared to the delays in the network connecting the datacenters. Thus, we will consider here just the delays in the inter-datacenter communications.

We consider a deterministic system with token bucket controlled traffic (as for TSPEC in IntServ) and rate-latency, as in [24]. We consider the delay accrued along the path traversed by chain $\mathbf{h}^u$'s traffic in the most general case, where such a path includes both uplink and downlink traffic transfers. To seamlessly model the downlink traffic, we denote by $\lambda^u_{h_u+1}$ the traffic from the last VM in the chain back to the PoA of the request.

Each link (i,j) is associated with propagation delay $T_p(i,j)$ and bandwidth capacity $C(i,j)$. We assume that on each link the bandwidth is partitioned between all the traversing flows, and a link scheduler provides a rate guarantee for each chain $\mathbf{h}^u$ equal to λ^u_1 for uplinks, and $\lambda^u_{h_u+1}$ for downlinks. We assume that the bandwidth on each link is sufficiently provisioned, thus $C(i,j)$ is large enough to accommodate all the traffic flows between i and j, thus avoiding blocking events.

We assume that ingress and egress chain traffic is leaky-bucket regulated, with maximum burstiness σ_u. The link (i,j) scheduler is a rate-latency server (e.g., a PGPS scheduler [24]) with the appropriate rate λ^u as specified above and latency $L_{\max}/C(i,j)$, where $L_{\max}$ is a constant depending upon the adopted scheduler (e.g., equal to the maximum packet size for PGPS). Using network calculus [24] and defining $\tau(i,j) = L_{\max}/C(i,j) + T_p(i,j)$, the delay experienced on link (i,j) can be upper-bounded by:

$$\frac{\sigma_u}{\lambda^u} + \tau(i,j)\,. \tag{3}$$

Then, recalling the "pay bursts only once" result [24], the *network delay* associated with chain $\mathbf{h}^u$ is:

$$d^n(\mathbf{h}^u, s) = \frac{\sigma_u}{\lambda^u_1} + \frac{\sigma_u}{\lambda^u_{h_u+1}} + \sum_{(i,j)\in\mathcal{P}(p^u,s)\cup\mathcal{P}(s,p^u)} \tau(i,j) \tag{4}$$

where s is the datacenter on which chain $\mathbf{h}^u$ is deployed.

Total service delay. The total delay of chain $\mathbf{h}^u$ is then given by the sum of its computational and network delay, i.e.,

$$d(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}, s) = d^c(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}) + d^n(\mathbf{h}^u, s). \tag{5}$$

III. The deployment and migration problem

The delay experienced by a chain may vary over time because (i) the PoA of the request, hence the network delay, has changed, or, (ii) there is a traffic surge/reduction, and the processing time of the chain VMs changes [25]. We assume that a monitoring system predicts the performance of the deployed services every T time units (hereinafter also referred to as the *decision period*), and it identifies the set of *critical* chains, whose experienced latency is expected to violate the delay constraints due to changes in the requests' attributes (e.g., PoA, or values of λ^u_k). The delay constraint of a critical chain may dictate migrating that chain to reduce its delay. Every decision period, the service orchestrator decides on the destination datacenter and on the resources to allocate for the new chains and for the critical chains that need to be migrated. Notably, according to our definition, $\mathcal{H}$ comprises the new chains, the critical chains, and the remaining currently deployed non-critical chains.

In what follows, we formulate an optimization problem defining the framework in which such decisions are made with the aim of minimizing the migration and the system operational cost. We first introduce the problem decision variables, constraints, system costs, and, then, our objective function. Finally, we discuss the problem complexity.

Decision variables. Let $\mathbf{y}$ denote the Boolean *placement* decision variables. Namely, $y(u,s) = 1$ iff chain $\mathbf{h}^u$ is scheduled to run on datacenter s in the following decision period. The *allocation* decision variables, $\boldsymbol{\mu}$, determine, for every VM v^u_k of chain $\mathbf{h}^u$, the amount of computing capacity $\mu^{u,s}_k$ to be allocated for this VM on datacenter s hosting the chain. Any choice for the values of the $\mathbf{y}$- and $\boldsymbol{\mu}$-variables comprises a *solution* to our problem, specifying (i) where new chains are deployed and what computing resources each of their VMs gets, and (ii) which existing chains are migrated, where they are migrated to, and what computing resources each of their VMs use.

Constraints. The following constraints hold:

$$\textstyle\sum_{s\in\mathcal{S}} y(u,s) = 1 \qquad \forall \mathbf{h}^u \in \mathcal{H} \tag{6}$$

$$y(u,s)\cdot d(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}, s) \leq \Delta(\mathbf{h}^u) \quad \forall \mathbf{h}^u \in \mathcal{H}, \forall s \in \mathcal{S} \tag{7}$$

$$y(u,s)\cdot \theta^u_k \cdot \lambda^u_k \leq \mu^{u,s}_k \qquad \forall \mathbf{h}^u \in \mathcal{H}, \forall s \in \mathcal{S} \tag{8}$$

$$\textstyle\sum_{k=1}^{h_u} \mu^{u,s}_k \leq \hat{C}_u \qquad \forall \mathbf{h}^u \in \mathcal{H} \tag{9}$$

$$\textstyle\sum_{\mathbf{h}^u\in\mathcal{H}} \sum_{k=1}^{h_u} \mu^{u,s}_k \leq C_s \qquad \forall s \in \mathcal{S}\,. \tag{10}$$

Indeed, (6) ensures that each chain is associated with a single *scheduled* placement. (7) guarantees that the target maximum delay of each chain is satisfied. (8) ensures a finite delay for each VM. (9) verifies the bound of computing resources allocated for each chain. Finally, (10) makes sure that the capacity of each datacenter is not exceeded.

Costs. The system costs are due to migration, as well as computation and bandwidth usage, as detailed below.

Migrating chain $\mathbf{h}^u$ from datacenter s to datacenter s' incurs a *computational migration cost* $\chi^m(u,s,s')$. Let $x(u,s)$ denote the *current placement indicator parameters*, i.e., $x(u,s) = 1$ iff chain $\mathbf{h}^u$ is currently placed on datacenter s.[2] The migration cost incurred by a critical chain $\mathbf{h}^u$ is then:

$$\sum_{s\neq s'\in\mathcal{S}} x(u,s)\cdot y(u,s')\cdot \chi^m(u,s,s').$$

Each unit of computation allocated on datacenter s incurs a *computation cost* $\chi^c(s)$. Finally, each unit of traffic being routed across link (i,j) incurs a *bandwidth cost* $\chi^b(i,j)$.

[2]Note that $x(u,s)$ are not decision variables to be determined, but rather represent the *current state* of the deployment.

Objective. Our objective is to minimize the cost function

$$\phi(\mathbf{y},\boldsymbol{\mu}) = \sum_{\mathbf{h}^u\in\mathcal{H}}\sum_{s\neq s'\in\mathcal{S}} x(u,s)\cdot y(u,s')\cdot\chi^m(u,s,s') + \sum_{\mathbf{h}^u\in\mathcal{H}}\sum_{s\in\mathcal{S}} y(u,s)\sum_{k=1}^{h_u}\mu_k^{u,s}\cdot\chi^c(s) + \sum_{(i,j)\in\mathcal{L}} b(i,j,\mathbf{y})\cdot\chi^b(i,j) \tag{11}$$

where $b(i,j,\mathbf{y})$ denotes the overall amount of traffic (in bit/s) that traverses link (i,j) when using placement $\mathbf{y}$. Namely, considering the traffic towards the datacenters and back:

$$b(i,j,\mathbf{y}) = \begin{cases} \sum_{\mathbf{h}^u\in\mathcal{H}} \sum_{\substack{s\in\mathcal{S}\\(i,j)\in\mathcal{P}(p^u,s)}} y(u,s)\cdot\lambda_1^u & \text{uplink} \\ \sum_{\mathbf{h}^u\in\mathcal{H}} \sum_{\substack{s\in\mathcal{S}\\(i,j)\in\mathcal{P}(s,p^u)}} y(u,s)\cdot\lambda_{h_u+1}^u & \text{downlink.} \end{cases} \tag{12}$$

Our problem, hereinafter referred to as the *Deployment and Migration Problem* (DMP), is therefore given by:

$$\min_{\mathbf{y},\boldsymbol{\mu}} \phi(\mathbf{y},\boldsymbol{\mu}) \text{ subject to } (6)-(10). \tag{13}$$

We can prove the following result on the DMP complexity.

Theorem 1. *The DMP is NP-hard.*

Proof: Consider the NP-hard partition problem [26], where we are given a sequence of integers $n_1,\ldots,n_k$, and we seek a set $N \subseteq \{1,\ldots,k\}$ such that $\sum_{i\in N} n_i = \left(\sum_{i=1}^k n_i\right)/2$. Without loss of generality, assume that $n_i \geq 2$ for all i (otherwise, we may simply consider the integers $\hat{n}_i = 2\cdot n_i$ for all i as our input).

We now present a polynomial reduction from the partition problem to the DMP. Consider a network with two datacenters, s and r, where r is the root, and s is co-located with the PoA of all requests. The processing capacity in both the root and the PoA is $C_s = C_r = \left(\sum_{i=1}^k n_i\right)/2$. Define requests $u = 1,\ldots,n$ where chain $\mathbf{h}^u$ has delay constraint $\Delta(\mathbf{h}^u) = \frac{1}{n_u-1}$. Each requested chain has a single VM, with rate $\lambda_1^u = \lambda_2^u = 1$, and requires a processing capacity $\theta_1^u = \theta_2^u = 1$. The network delay is zero. Observe that the delay constraint (7) of chain i is satisfied only if it is allocated at least n_i CPU units. Furthermore, since $C_s + C_r = \sum_{i=1}^k n_i$, a feasible solution for DMP may allocate for chain i at most n_i CPU units. It follows that any feasible solution allocates exactly n_i CPU units for chain i. Hence, there exists a feasible solution for DMP for this input iff there exists a solution to the partition problem. The result follows. ∎

IV. Solution overview and main results

The DMP's objective (11) combines the placement decision variables, $\mathbf{y}$, and the allocation decision variables, $\boldsymbol{\mu}$. A closer look shows that the chain placement and CPU allocation problems are entangled, since each placement decision impacts the CPU allocation required to satisfy the target delay of the service. Our solution concept is based on *decoupling the chain placement and CPU allocation problems*, which allows applying a combinatorial approach to solving the DMP, and studying the trade-offs inherent to our solutions. In more detail, our solution comprises three steps: (i) solving the CPU allocation problem (Sec. V), (ii) finding a feasible solution for the chain placement problem (Sec. VI), and (iii) reducing cost (Sec. VII). We now overview these steps.

TABLE I: Main notations

Symbol	Description
Parameters: network (Sec. II-A)	
$\mathcal{G}$	Network graph
$D(\mathcal{G})$	The diameter of network graph $\mathcal{G}$
$\mathcal{S}$	Set of datacenters
$\mathcal{L}$	Set of links
(i,j)	Directed link connecting datacenters i and j
$\mathcal{P}(s,s')$	Path connecting datacenter s to datacenter s'
Parameters: services, delays and capacities (Secs. II-B–II-C)	
$\mathcal{U}$	Set of service requests
$\mathcal{H}$	Set of chains corresponding to $\mathcal{U}$
$\mathbf{h}^u$	Service chain (ordered list of VMs) serving u
h_u	Number of VMs in $\mathbf{h}^u$
p^u	PoA where request for $\mathbf{h}^u$ is generated
$\Delta(\mathbf{h}^u)$	Target delay [s] of chain $\mathbf{h}^u$
$\hat{C}_u$	Maximum CPU units that may be allocated to chain $\mathbf{h}^u$ based on the SLA
C_s	Overall processing capacity of datacenter s [cycles/s]
λ_k^u	Input traffic load of VM v_k^u [bits/sec]
θ_k^u	Required processing capacity for VM v_k^u's incoming traffic [cycles / bit]
γ_k^u	Bits per data units [bit]
$D_k^u(\mu_k^{u,s})$	Computational delay [s] exhibited by VM v_k^u (1)
$d^c(\mathbf{h}^u,\boldsymbol{\mu}^{u,s})$	Computational delay [s] of chain $\mathbf{h}^u$ (2)
$\tau(i,j)$	Network delay experienced on link (i,j)
$d^n(\mathbf{h}^u,s)$	Network delay [s] of chain $\mathbf{h}^u$ when located on server s (4)
Parameters: costs (Sec. III)	
$\chi^m(u,s,s')$	Cost of migrating chain $\mathbf{h}^u$ from datacenter s to datacenter s'
$x(u,s)$	Placement indicator: true iff chain $\mathbf{h}^u$ is currently hosted on datacenter s
$\chi^c(s)$	Cost of allocating one CPU unit on datacenter s
$\chi^b(i,j)$	Cost of having one unit of bandwidth traverse link (i,j)
ϕ	Objective function (11)
$b(i,j,\mathbf{y})$	Amount of traffic traversing link (i,j) [bits/s] (12)
Decision variables (Sec. III)	
$y(u,s)$	Placement indicator: true iff chain $\mathbf{h}^u$ is scheduled to run on datacenter s
$\mu_k^{u,s}$	Integer allocation: expressing the number of CPU units allocated for VM v_k^u on datacenter s
Sec. V	
$\mathcal{S}_u$	Set of delay-feasible datacenters of chain $\mathbf{h}^u$
$\mu_k^{u,s}$	CPU allocation for the k-th VM of chain $\mathbf{h}^u$ on datacenter s
δ_k^u	Delay reduction function (14)
Sec. VI	
$\mathcal{T}(s)$	The sub-tree rooted by datacenter s
$\overline{s}(u)$	Top datacenter in $\mathcal{S}_u$
$\mathcal{H}(s)$	The set of chains whose PoAs are in $\mathcal{T}(s)$
$\tilde{\mu}(u)$	The minimum required CPU to serve request u as in (28)
$T_{\mathcal{H}'}$	Potential placement tree of $\mathcal{H}'$
R	Multiplicative resource augmentation factor
a_s	Available processing capacity of datacenter s

A. Solving the CPU allocation problem

In Sec. V, we define a polynomial-time algorithm, called *GetFeasibleAllocations* (GFA), that identifies for each chain $\mathbf{h}^u$ its set of *delay-feasible* datacenters, namely, the datacenters on which it is possible to place $\mathbf{h}^u$, while meeting its target delay. For each chain $\mathbf{h}^u$ and delay-feasible datacenter s, GFA finds an allocation $\boldsymbol{\mu}^{u,s}$ that is *provably minimal* in terms of the overall number of required CPU units.

B. Solving the chain placement problem

First, we note that given any allocation $\boldsymbol{\mu}^{u,s}$ for all chains $\mathbf{h}^u$ and datacenters s, the DMP (13) becomes an integer linear

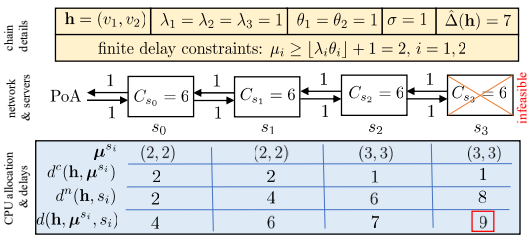


Fig. 2: Example of the resource allocation dynamics and their effect on delay and feasibility. The input chain details are given at the top; the datacenter topology is given in the middle, with all link delays and datacenter capacities set to 1 and 6 (resp.). The table at the bottom indicates: (i) the minimal allocation $\boldsymbol{\mu}^{s_i}$ on datacenter s_i, (ii) the induced computational delay $d^c(\mathbf{h}, \boldsymbol{\mu}^{s_i})$, (iii) the network delay incurred for reaching datacenter s_i, and (iv) the total delay $d(\mathbf{h}, \boldsymbol{\mu}^{s_i}, s_i)$. As can be seen, while datacenters s_0, s_1, s_2 are feasible (each with its CPU allocation), datacenter s_3 is infeasible.

program (ILP). We note that the optimal solution for the linear relaxation serves a lower bound for the DMP, and also serves as a witness of feasibility.

The proof of Theorem 1 implies that even when the solution for the CPU allocation problem is known (e.g., allocating n_i CPU units to VM i in the proof of Theorem 1), the DMP is NP-hard. Hence, the following proposition holds.

Proposition 2. *Finding a feasible solution to the chain placement problem is NP-hard.*

In Sec. VI, we address the hardness of the chain placement problem using *resource augmentation*, i.e., assuming that each datacenter has an augmented processing capacity. We develop a polynomial-time algorithm, dubbed *Bottom-Up (BU)*. Further, we show an upper bound on the amount of processing capacity augmentation required for BU to find a feasible solution whenever one exists for the non-augmented case.

C. Reducing cost

In Sec. VII we present the *Push-Up (PU)* algorithm, which aims at reducing the cost of any given feasible DMP solution. Then, we use BU and PU as building blocks of our integrated algorithm *Bottom-Up-and-Push-Up (BUPU)* for finding a minimal-cost solution to DMP while minimizing the amount of resource augmentation.

V. Allocating Computational Resources

In this section, we address the CPU *allocation problem*. In particular, for each chain $\mathbf{h}^u$, we identify its set of *delay-feasible* datacenters, denoted by $\mathcal{S}_u$. Then, for every chain $\mathbf{h}^u$ and datacenter $s \in \mathcal{S}_u$, we calculate the minimal number of CPU units that one must assign to each VM in $\mathbf{h}^u$ running on s, in order to satisfy the target delay. Fig. 2 provides some intuition on the allocation problem.

Algorithm 1 GFA$(\mathcal{G}, \tilde{\mathcal{H}})$

1: **for** $\mathbf{h}^u \in \tilde{\mathcal{H}}$ **do** ▷ for each chain
2: $\mathcal{S}_u = \mathcal{P}(p^u, r)$ ▷ ordered list of datacenters from PoA to the root
3: **for each** $s \in \mathcal{S}_u$ **do** ▷ for each datacenter in $\mathcal{S}_u$, from PoA to the root
4: **for** $k = 1, \ldots, h_u$ **do** ▷ for each chain in the VM
5: $\mu_k^{u,s} = \lfloor \theta_k^u \lambda_k^u \rfloor + 1$ ▷ ensure finite computation delay
6: **while** $d^c(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}) > \Delta(\mathbf{h}^u) - d^n(\mathbf{h}^u, s)$ **and** $\sum_{k=1}^{h_u} \mu_k^{u,s} \leq \hat{C}_u$ **do** ▷ delay constraint is still unsatisfied and CPU is available
7: $k^* = \arg\max_{1 \leq k \leq h_u} \{\delta_k^u(\mu_k^{u,s})\}$ ▷ find VM with max del. reduction
8: $\mu_{k^*}^{u,s} = \mu_{k^*}^{u,s} + 1$ ▷ increase CPU to reduce delays
9: **if** $\sum_{k=1}^{h_u} \mu_k^{u,s} > \hat{C}_u$ **then** ▷ if not enough CPU capacity, s is infeasible
10: $\mathcal{S}_u = \mathcal{S}_u \setminus \mathcal{P}(s, r)$ ▷ remove all the datacenters from s to the root
11: **break** ▷ it is not worth anymore to go further towards the root
12: **return** $\mathcal{S}_u, \forall \mathbf{h}^u \in \tilde{\mathcal{H}}$ and $\boldsymbol{\mu}^{u,s}, \forall \mathbf{h}^u \in \tilde{\mathcal{H}}, s \in \mathcal{S}$

A. The CPU allocation algorithm

Our polynomial-time CPU allocation algorithm, named GetFeasibleAllocations (or GFA, for short), takes as input the network graph $\mathcal{G}$ and a given set of chains $\tilde{\mathcal{H}}$. GFA computes the set of delay-feasible datacenters $\mathcal{S}_u$ for each chain $\mathbf{h}^u \in \tilde{\mathcal{H}}$, and, for each such datacenter s, calculates a CPU allocation vector $\boldsymbol{\mu}^{u,s}$ satisfying the delay constraint and minimizing the CPU allocated to $\mathbf{h}^u$ on s. Formally, $\boldsymbol{\mu}^{u,s} \in \arg\min_{\boldsymbol{\mu}} \{\|\boldsymbol{\mu}\|_1 \mid d(\mathbf{h}^u, \boldsymbol{\mu}) \leq \Delta_s(\mathbf{h}^u)\}$, where $\|\cdot\|_1$ is the ℓ_1 norm; such a CPU allocation is referred to as *minimal allocation*.

GFA is detailed in Alg. 1. For each chain $\mathbf{h}^u \in \tilde{\mathcal{H}}$, initially all datacenters in $\mathcal{P}(p^u, s)$ are assumed to be feasible (ln. 2). For each feasible datacenter, going from the PoA towards the root of the network, GFA initializes the CPU allocation for each VM in $\mathbf{h}^u$ to the minimal necessary to ensure a finite computational delay (ln. 4–5). The algorithm then computes (ln. 6-8) the minimum amount of CPU required to meet the delay constraint, while not violating the bound (9) on the computing resources allocated to the chain. This is done using the method described below. Finally, once the current datacenter s is delay infeasible, which by our model also implies that all its ancestors are deemed infeasible, s and all its ancestors are removed from the set of feasible datacenters (ln. 9-11), and GFA returns the set of feasible datacenters, and the corresponding minimal allocations (ln. 12).

Computing the minimal allocation. If the current allocation $\boldsymbol{\mu}$ leads to a delay constraint violation, GFA increases the total number of CPU units it uses by one using a gradient method: it increments the number of CPU units allocated to the VM that, owing to this change, maximizes the delay reduction (ln. 7–8). To this end, we define the *delay reduction function*, which captures the residual reduction in the computational delay corresponding to increasing the CPU allocation of v_j^u by one. Formally, for VM v_k^u and CPU allocation μ, we have:

$$\delta_k^u(\mu) = D_k^u(\mu) - D_k^u(\mu + 1)\,. \tag{14}$$

As can be verified by algebraic manipulation, $\delta_k^u(\mu)$ is monotonically decreasing for every $\mu > \lfloor \theta_k^u \lambda_k^u \rfloor$. Next, we prove that our approach indeed finds a minimal CPU allocation.

B. Performance analysis

We begin by defining the B-*minimal* CPU allocation for a given chain and CPU budget.

Definition 3. *A CPU allocation for chain $\mathbf{h}^u$ on datacenter s is B-minimal for a given CPU budget B, if it minimizes the computational delay of $\mathbf{h}^u$ on s while using B CPU units, i.e.,*

$$\tilde{\boldsymbol{\mu}}^{u,s}(\mathbf{h}^u, B) = \arg\min_{\boldsymbol{\mu}} \{d^c(\mathbf{h}^u, \boldsymbol{\mu}^{u,s}) | \, \|\boldsymbol{\mu}^{u,s}\|_1 = B\}. \quad (15)$$

To prove that GFA finds a minimal CPU allocation, we will use the following lemma on B-minimal CPU allocations.

Lemma 4. *Let $\boldsymbol{\mu}^*$ be a B-minimal allocation for chain $\mathbf{h}^u$ on datacenter s. Then, for any $1 \leq i, j \leq h_u$: $\delta_j^u(\mu_j^* - 1) \geq \delta_i^u(\mu_i^*)$.*

Proof. Assume by contradiction that

$$\begin{aligned}\delta_j^u(\mu_j^* - 1) &= D_j^u(\mu_j^* - 1) - D_j^u(\mu_j^*) \\ &< D_i^u(\mu_i^*) - D_j^u(\mu_i^* + 1) = \delta_i^u(\mu_i^*). \quad (16)\end{aligned}$$

This implies that $D_j^u(\mu_j^* - 1) + D_j^u(\mu_i^* + 1) < D_j^u(\mu_j^*) + D_i^u(\mu_i^*)$. Consider the allocation $\boldsymbol{\mu}'$, defined as: $\mu_j' = \mu_j^* - 1$, $\mu_i' = \mu_i^* + 1$, and for any $k \neq i, j$: $\mu_k' = \mu_k^*$. Thus, $d^c(\mathbf{h}^u, \boldsymbol{\mu}') - d^c(\mathbf{h}^u, \boldsymbol{\mu}^*) < 0$, which contradicts the B-minimality of $\boldsymbol{\mu}^*$. □

The following lemma shows that GFA never increases the CPU allocation to a VM above some level, before it exploits any chance to gain more delay reduction by increasing the CPU allocated to any other VM in that chain.

Lemma 5. *If $\delta_j^u(a) > \delta_i^u(b)$, GFA does not assign more than b CPU units to v_i^u before assigning at least $a+1$ units to v_j^u.*

Proof. If $a \leq \lfloor \theta_k^u \lambda_k^u \rfloor$, GFA initializes $\mu_j^{u,s}$ to at least $a+1$ units (ln. 5), and the claim holds true in the first iteration of the while loop. For any subsequent iteration, the above lemma holds by construction (ln. 7–8) since GFA will not assign $k^* = i$ (since it picks the VM that maximizes the improvement) before v_j^u is assigned at least $a+1$ CPU units. Note that the inequality $\delta_j^u(a) > \delta_i^u(b)$ is independent of any change made to the allocation of CPU units to any VM distinct from both i and j. □

The following lemma bounds the ℓ_∞ distance between the allocations that GFA considers, and any minimal allocation using an identical budget.

Lemma 6. *Let $\boldsymbol{\mu}$ be the allocation for chain $\mathbf{h}^u$ on datacenter s in some iteration of GFA's* while *loop, and let $B = \sum_{k=1}^{h_u} \mu_k^{u,s}$. Let $\boldsymbol{\mu}^*$ denote a B-minimal allocation for $\mathbf{h}^u$ on datacenter s. Then, $\|\boldsymbol{\mu} - \boldsymbol{\mu}^*\|_\infty \leq 1$.*

Proof. Assume by contradiction that the $\|\boldsymbol{\mu} - \boldsymbol{\mu}^*\|_\infty > 1$. Then, there exists an index i for which either (1) $\mu_i - 1 > \mu_i^*$, or (2) $\mu_i^* - 1 > \mu_i$. We have now two cases.

Case 1: $\mu_i - 1 > \mu_i^*$. As $B = \sum_{k=1}^{h_u} \mu_k = \sum_{k=1}^{h_u} \mu_k^*$, there exists another index $j \neq i$ s.t. $\mu_j^* - 1 \geq \mu_j$. Namely,

$$\mu_i - 1 > \mu_i^* \quad , \quad \mu_j^* - 1 \geq \mu_j. \quad (17)$$

By Lemma 4,

$$\delta_j^u(\mu_j^* - 1) \geq \delta_i^u(\mu_i^*). \quad (18)$$

Combining (17) and the fact that $\delta^u(\cdot)$ is monotone decreasing, we have that $\delta_i^u(\mu_i^*) > \delta_i^u(\mu_i - 1)$, $\delta_j^u(\mu_j) \geq \delta_j^u(\mu_j^* - 1)$. Combining these inequalities with (18), we have $\delta_j^u(\mu_j) > \delta_i^u(\mu_i - 1)$. However, applying Lemma 5 on this latter inequality implies that GFA does not assign μ_i CPU units for v_i^u before assigning at least $\mu_j + 1$ CPU units to v_j^u. Hence, this case is impossible.

Case 2: $\mu_i^* - 1 > \mu_i$. As $B = \sum_{k=1}^{h_u} \mu_k = \sum_{k=1}^{h_u} \mu_k^*$, there exists an index $j \neq i$ s.t. $\mu_j - 1 \geq \mu_j^*$. Namely,

$$\mu_i^* - 1 > \mu_i \quad , \quad \mu_j - 1 \geq \mu_j^*. \quad (19)$$

Applying Lemma 4, while exchanging the roles of i and j, we have $\delta_i^u(\mu_i^* - 1) \geq \delta_j^u(\mu_j^*)$. Combining (19) and the fact that $\delta^u(\cdot)$ is monotone decreasing, we obtain $\delta_i^u(\mu_i) > \delta_i^u(\mu_i^* - 1)$. Combining the latter two inequalities, we have

$$\delta_i^u(\mu_i) > \delta_j^u(\mu_j^*). \quad (20)$$

By setting $a = \mu_i$ and $b = \mu_j^*$ in Lemma 5, we know that GFA does not assign v_j^u more than μ_j^* units before assigning to v_i^u at least $\mu_i + 1$ units. However, this contradicts our assumption that vector $\boldsymbol{\mu}$ assigns only μ_i CPU units to v_i^u, while v_j^u is already allocated some $\mu_j > \mu_j^*$ units. Therefore, also case 2 is impossible, and the thesis follows. □

The following lemma shows that GFA considers only B-minimal allocations.

Lemma 7. *Let $\boldsymbol{\mu}$ be the allocation in some iteration of GFA's* while *loop for chain $\mathbf{h}^u$ on datacenter s, and let $B = \|\boldsymbol{\mu}\|_1$. Then, $\boldsymbol{\mu}$ is B-minimal.*

Proof. Let $\boldsymbol{\mu}^*$ denote a B-minimal allocation for chain $\mathbf{h}^u$ on datacenter s, and assume by contradiction that $\boldsymbol{\mu}$ is not B-minimal. By Lemma 6, for any $1 \leq i \leq h_u : |\mu_i - \mu_i^*| \leq 1$. Then we can partition the non-equal indices in $\boldsymbol{\mu}$ and $\boldsymbol{\mu}^*$ into K pairs, where pair k consists of two indices $1 \leq i_k, j_k \leq h_u$ s.t.

$$\mu_{i_k}^* = \mu_{i_k} - 1, \quad \mu_{j_k}^* = \mu_{j_k} + 1. \quad (21)$$

Applying Lemma 4 with $i = i_k$ and $j = j_k$, we have $\delta_{j_k}^u(\mu_{j_k}^* - 1) \geq \delta_{i_k}^u(\mu_{i_k}^*)$. Combining this with (21), we have

$$\delta_{j_k}^u(\mu_{j_k}) = \delta_{j_k}^u(\mu_{j_k}^* - 1) \geq \delta_{i_k}^u(\mu_{i_k}^*) = \delta_{i_k}^u(\mu_{i_k} - 1). \quad (22)$$

Eq. (22) implies that either there exists k s.t. $\delta_{j_k}^u(\mu_{j_k}) > \delta_{i_k}^u(\mu_{i_k} - 1)$, or for all k, it holds that $\delta_{j_k}^u(\mu_{j_k}) = \delta_{i_k}^u(\mu_{i_k} - 1)$. In the former case, by assigning $a = \mu_{j_k}$ and $b = \mu_{i_k} - 1$ in Lemma 5, we know that GFA will not assign μ_{i_k} CPU units to v_i^u before assigning at least $\mu_{j_k} + 1$ CPU units to v_j^u. Since the current allocation is μ_i units to v_i^u, and μ_j units to v_j^u, a contradiction arises. Thus, we have that for all k,

$$\delta_{j_k}^u(\mu_{j_k}) = \delta_{i_k}^u(\mu_{i_k} - 1). \quad (23)$$

We now show that if (23) holds, then the total delay obtained by $\boldsymbol{\mu}$ equals that obtained by $\boldsymbol{\mu}^*$, thus contradicting the assumption that $\boldsymbol{\mu}$ is not minimal.

By the definition of $\delta(\cdot)$ in (14), we get

$$\delta_{i_k}^u(\mu_{i_k} - 1) = D_{i_k}^u(\mu_{i_k} - 1) - D_{i_k}^u(\mu_{i_k}) = D_{i_k}^u(\mu_{i_k}^*) - D_{i_k}^u(\mu_{i_k}), \quad (24)$$

where the latter equation is by (21). Similarly, we have

$$\delta_{j_k}^u(\mu_{j_k}) = D_{j_k}^u(\mu_{j_k}) - D_{j_k}^u(\mu_{j_k} + 1) = D_{j_k}^u(\mu_{j_k}) - D_{j_k}^u(\mu_{j_k}^*), \quad (25)$$

Combining (23), (24) and (25), we obtain:

$$D_{i_k}^u(\mu_{i_k}) - D_{i_k}^u(\mu_{i_k}^*) + D_{j_k}^u(\mu_{j_k}) - D_{j_k}^u(\mu_{j_k}^*) = 0. \quad (26)$$

By definition of K, the difference in computational delay due to $\boldsymbol{\mu}$ and $\boldsymbol{\mu}^*$ is

$$d^c(\mathbf{h}^u, \boldsymbol{\mu}) - d^c(\mathbf{h}^u, \boldsymbol{\mu}^*) = \sum_{i=1}^{h_u} \left[D_i^u(\mu_i) - D_i^u(\mu_i^*) \right] = \sum_{k=1}^{K} \left[D_{i_k}^u(\mu_{i_k}) - D_{i_k}^u(\mu_{i_k}^*) + D_{j_k}^u(\mu_{j_k}) - D_{j_k}^u(\mu_{j_k}^*) \right] = 0. \tag{27}$$

By (27), the delay due to $\boldsymbol{\mu}$ is the same as that due to $\boldsymbol{\mu}^*$, thus contradicting our assumption on $\boldsymbol{\mu}$ not being B-minimal. □

By Lemma 7, the following corollary holds.

Corollary 8. *For every chain $\mathbf{h}^u$ and datacenter s, if there exists a feasible allocation for $\mathbf{h}^u$ on s, then the allocation $\boldsymbol{\mu}^{u,s}$ given by GFA satisfies $\boldsymbol{\mu}^{u,s} \in \arg\min_{\boldsymbol{\mu}} \{ \|\boldsymbol{\mu}\|_1 \mid d^c(\mathbf{h}^u, \boldsymbol{\mu}) \leq \Delta_s(\mathbf{h}^u) \}$, i.e., it minimizes the number of allocated CPU units over all feasible allocations for $\mathbf{h}^u$ on s.*

Proof. As GFA increments the total used CPU budget by one at each iteration (ln. 8), we know that the budget that GFA used in the previous iteration, if exists, was $B-1$. By Lemma 7, no other allocation obtains lower delay for chain $\mathbf{h}^u$ with budget $B-1$. Hence, no allocation satisfies the target delay constraint using budget $B-1$. □

Run-time analysis: The time complexity of allocating CPU to $\mathbf{h}^u$ on s is $\tilde{O}(\hat{C}_u + h_u)$. For each of the $\mathcal{H}$ chains, GFA considers at most $D(\mathcal{G})$ possible servers. Thus, the overall time complexity of running GFA is $\tilde{O}(|\mathcal{H}| \cdot D(\mathcal{G}) \cdot \max_{\mathbf{h}^u \in \mathcal{H}} (\hat{C}_u + h_u))$.

VI. Feasible solution to the placement problem

In light of the complexity of the placement problem (see Proposition 2), we now introduce the BU algorithm, which finds a feasible solution to chain placement with some resource augmentation. In particular, after introducing some preliminaries, we prove that, using some bounded resource augmentation, BU always finds a feasible solution if such a solution exists in the non-augmented case.

A. Preliminaries

Let $\tilde{\mu}$ be a lower bound on the number of CPU units required to successfully serve any chain instance. By Corollary 8, $\tilde{\mu}$ can be computed using the allocations $\boldsymbol{\mu}$ found by GFA:

$$\tilde{\mu} = \min_{s \in \mathcal{S}, u \in \mathcal{U}} \|\boldsymbol{\mu}^{u,s}\|_1 . \tag{28}$$

Let $\mathcal{T}(s)$ denote the sub-tree rooted at datacenter s. Further, denote by $\mathcal{H}(s)$ the set of chains whose PoAs are in $\mathcal{T}(s)$. Denote by $\overline{s}(u)$ the top datacenter in $\mathcal{S}_u$ (i.e., the farthest datacenter from PoA p^u that is delay-feasible for $\mathbf{h}^u$). Given a set of chains $\mathcal{H}' \subseteq \mathcal{H}$, we define the *potential placement tree* of $\mathcal{H}'$ as

$$\mathcal{T}_{\mathcal{H}'} = \bigcup_{\mathbf{h}^u \in \mathcal{H}'} \mathcal{S}_u. \tag{29}$$

An illustration clarifying the above notation is provided in Fig. 3.

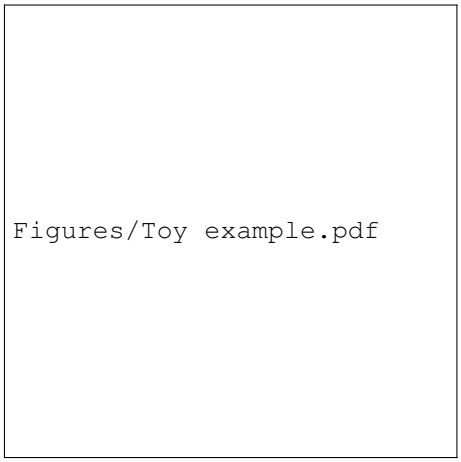


Fig. 3: Example of placement tree. The PoA for requests u_1, u_2, and u_3 is s_5, while that for requests u_4, u_5, and u_6 is s_3. For each request, the top datacenter of the relevant chain is denoted by $\overline{s}(u)$. The potential placement trees of $\{u_1, u_2, u_3\}$ and of $\{u_4, , u_5, u_6\}$ are highlighted in yellow and blue (resp.), while their union ($\mathcal{T}(s_0)$) is that of $\{u_1, u_2, u_3, u_4, u_5, u_6\}$.

B. Motivation for a Bottom-Up approach

When one looks at the problem of minimizing the placement cost, it may seem prudent to try and place chains as high as possible. Indeed, processing costs are lower the farther the datacenter is from the chain's PoA. For instance, running a VM in a datacenter residing in the cloud is much cheaper than running it at a MEC datacenter attached to the PoA, where capacity is scarce and expensive. Such an approach can be combined with a "back-pressure" mechanism, which tries to push previously placed chains lower in the hierarchy, whenever a request cannot be placed at its highest delay-feasible datacenter. However, when targeting a feasible placement, such a mechanism might end up performing an exhaustive search.

To see this, consider a scenario where a chain $\mathbf{h}^u$, originating at PoA p^u, cannot be accommodated in any of the datacenters along the sequence $\mathcal{S}_u$, due to previously placed chains. In such a case, there might be a *unique* sequence of chains $\mathbf{h}^{u_0}, \ldots, \mathbf{h}^{u_t}$ and a sequence of datacenters $s_0, \ldots s_{t+1}$, such that (i) $s_0 \in \mathcal{S}_u$, (ii) $\mathbf{h}^{u_i}$ is currently placed in s_i, for $i = 1, \ldots, t$, and (iii) re-placing $\mathbf{h}^{u_i}$ in s_{i+1} for $i = 1, \ldots, t$, and placing h_u in s_0 is feasible (given the placement of all other chains already handled). Since there is no clear criteria for identifying such a sequence of chains and datacenters, simply testing all back-pressure adjustments may be prohibitively costly. Since such a sequence may be unique (i.e., no other sequence can meet the above requirements), performing such an exhaustive search might be *necessary* to find a feasible solution.

The above scenario serves as a motivation for our *Bottom-Up (BU) algorithm* described in the sequel, which, by design, avoids such predicaments altogether. We stress that the BU algorithm is meant to find a feasible solution efficiently. Once such a solution is found, we enhance it by pushing-up chains to decrease the total cost, as detailed in Sec. VII.

Algorithm 2 BU $(\mathcal{G}, \tilde{\mathcal{H}}, \{\boldsymbol{\mu}^{u,s}\}_{u\in\mathcal{U},s\in\mathcal{S}}, \{\mathcal{S}_u\}_{u\in\mathcal{U}}, \mathbf{a})$

1: $\mathbf{a} \leftarrow \mathbf{a}$ after releasing resources of chains in $\tilde{\mathcal{H}}$ ▷ Release resources
2: $\mathbf{y} \equiv 0$ ▷ Init placement
3: **for each** datacenter s in DFS order **do**
4: **for** $\mathbf{h}^u \in \tilde{\mathcal{H}} \cap \mathcal{H}(s)$ in non-decreasing order of $|\mathcal{S}_u \setminus \mathcal{T}(s)|$ **do**
5: **if** $a_s \geq \|\boldsymbol{\mu}^{u,s}\|_1$ **then** ▷ Check if enough residual capacity
6: $y(u,s) = 1$ ▷ Deploy the chain
7: $a_s = a_s - \|\boldsymbol{\mu}^{u,s}\|_1$ ▷ Update the residual capacity
8: **else if** $\overline{s}(u) = s$ **then** ▷ Check if the top datacenter has been reached
9: return $\mathbf{y} \equiv 0, \mathbf{a} \equiv 0$ ▷ Infeasible
10: return $\mathbf{y}$, $\mathbf{a}$

C. The BU algorithm

The BU algorithm tries to place every chain as *low* (namely, closest to the corresponding PoA) as possible, and climbs higher only when there is insufficient processing capacity in any of the lower levels datacenters. While this approach may seem a poor choice in terms of cost, it is effective in ensuring a feasible placement. Intuitively, if the BU algorithm fails to place some chain, then there exists a sub-tree that is overloaded with service requests, so that other algorithms are also unlikely to find a feasible solution. Our algorithm will be using an *augmented processing capacity*, namely, BU may allocate on datacenter s up to $R \cdot C_s$ CPU units, where $R \geq 1$ is the multiplicative resource augmentation factor. We denote by a_s the currently available (i.e., residual) processing capacity in datacenter s during the execution of BU.

BU is detailed in Alg. 2. It gets as input the network graph $\mathcal{G}$; a set $\tilde{\mathcal{H}}$ of chains to be placed or migrated; the CPU allocation and the set of delay-feasible datacenters of each chain; and the currently available processing capacity $\mathbf{a}$. Given the above inputs, BU computes placement $\mathbf{y}$ and the new residual available processing capacity $\mathbf{a}$. BU first releases the resources of all the chains, that will potentially be re-placed (ln. 1). Then, BU scans the datacenters in a Depth First Search (DFS) order, and for each datacenter s in this scan, it tries to deploy yet unplaced chains on s, while satisfying the (residual) capacity constraint on s (ln. 5). The yet unplaced chains are scanned in such a way that chains $\mathbf{h}^u$ for which there are fewer remaining placement options are considered first. The number of remaining placement options is the number of datacenters that are delay-feasible for $\mathbf{h}^u$ but found above s, namely $|\mathcal{S}_u \setminus \mathcal{T}(s)|$. In case a chain cannot be placed on any of its feasible datacenters, BU returns an empty placement, which serves as a signal that the problem is infeasible (as proved in the sequel).

D. Ensuring feasibility

We now upper-bound the amount of processing capacity resource augmentation, henceforth R, that guarantees that BU finds a feasible solution, if such a solution exists in the case with no resource augmentation. Intuitively, our proof shows that if BU fails, then there exists a sub-tree $\mathcal{T}_{\mathcal{H}'}$ that is overloaded by some set of chains $\mathcal{H}'$ that must be placed on $\mathcal{T}_{\mathcal{H}'}$. Further, we show that in such a case *any* algorithm that does not use augmented processing capacities will fail as well.

We begin by characterizing the case where BU may fail to place a chain on some datacenter s.

Definition 9. *A datacenter s is almost-full if $a_s < \max_{u\in\mathcal{U}} \hat{C}_u$, and a set of datacenters $\mathcal{S}'$ is almost-full if every $s \in \mathcal{S}'$ is almost-full.*

The notion of an almost-full datacenter helps to upper-bound the amount of resource augmentation used by BU. Further, our evaluation study in Sec. VIII shows that the amount of resource augmentation used by BU is significantly lower than our worst-case guarantees. The following lemma provides a lower bound on the number of chains that BU places on a datacenter before it becomes almost-full.

Lemma 10. *If s is almost-full, then BU has placed on s at least $\lfloor R \cdot C_s / \max_{u\in\mathcal{U}} \hat{C}_u \rfloor$ chains.*

Proof. The augmented processing capacity on datacenter s is $R \cdot C_s$. Now BU allocates at most $\max_{u\in\mathcal{U}} \hat{C}_u$ CPU units per chain. The result follows. □

In the sequel we assume that $R \cdot C_s / \max_{u\in\mathcal{U}} \hat{C}_u$ is an integer. The following claim follows from the fact that BU attempts to deploy any unplaced chain $\mathbf{h}^u$ as close as possible to p^u.

Lemma 11. *If BU tries to place chain $\mathbf{h}^u$ on datacenter s, then all the datacenters in $\mathcal{S}_u$ below s are almost-full.*

The following lemma follows directly from the order in which unplaced chains are considered at any datacenter.

Lemma 12. *Consider two chains, $\mathbf{h}^u$, $\mathbf{h}^{u'}$, unplaced when considering datacenter $s \in \mathcal{S}_u \cap \mathcal{S}_{u'}$. Assume that (i) BU places $\mathbf{h}^{u'}$ on s before it tries to place $\mathbf{h}^u$ on s, and (ii) $\mathcal{S}_u$ is almost-full after BU terminates. Then $\mathcal{S}_{u'}$ is also almost-full after BU terminates.*

Proof. Let s' be a datacenter in $\mathcal{S}_{u'}$. We will show that s' is almost-full. Consider three cases, corresponding to s' being a descendent of s, an ancestor of s, or equals to s.

Case 1: s' is a descendent of s. By condition (i), s places $\mathbf{h}^{u'}$. Hence, by Lemma 11, all the datacenters in $\mathcal{S}_{u'}$ below s are almost-full. It follows that s' is almost-full.

Case 2: s' is an ancestor of s. By condition (i) and the order in which BU considers the chains (ln. 4 in Alg. 2), we know that $|\mathcal{S}_{u'} \setminus \mathcal{T}(s)| \leq |\mathcal{S}_u \setminus \mathcal{T}(s)|$. In addition, $s \in \mathcal{S}_u \cap \mathcal{S}_{u'}$. Combining the reasoning above, every ancestor of s belonging to $\mathcal{S}_{u'}$ belongs also to $\mathcal{S}_u$. In particular, s' is an ancestor of s belonging to $\mathcal{S}_{u'}$, and therefore $s' \in \mathcal{S}_u$. As $\mathcal{S}_u$ is almost-full, s' is almost-full.

Case 3: $s' = s$. As it is given that $s \in \mathcal{S}_u$ and $\mathcal{S}_u$ is almost-full, s' is almost-full. □

The following lemma shows that if BU fails, then there exists a set of chains requiring a total amount of CPU resources that is higher than the overall (augmented) processing capacity of the relevant datacenters.

Lemma 13. *If BU fails to place a chain, then there exists a set of chains $\mathcal{H}'$ s.t.*

$$|\mathcal{H}'| > \frac{R}{\max_{u\in\mathcal{U}} \hat{C}_u} \sum_{s\in T_{\mathcal{H}'}} C_s \,. \tag{30}$$

Algorithm 3 Constructing $\mathcal{H}'$ satisfying (30)

1: $\mathbf{h}^{\tilde{u}}$ = chain that BU failed to place, unmarked
2: $\mathcal{H}' = \left\{\mathbf{h}^{\tilde{u}}\right\}$
3: **for each** unmarked chain $\mathbf{h}^u \in \mathcal{H}'$ **do**
4: mark $\mathbf{h}^u$
5: **for each** un-visited datacenter s from $\overline{s}(u)$ to p^u **do**
6: mark s as visited
7: **for each** unmarked chain $\mathbf{h}^{u'}$ placed on s **do**
▷ w.l.o.g., in reverse order of placement by BU
8: add $\mathbf{h}^{u'}$ to $\mathcal{H}'$, unmarked

Proof. Let $\mathbf{h}^{\tilde{u}}$ be the chain that BU fails to place. Our construction of $\mathcal{H}'$ is detailed in Alg. 3, which works as follows. First, it initializes $\mathcal{H}'$ to $\left\{\mathbf{h}^{\tilde{u}}\right\}$; then, it repeatedly visits all the datacenters that are delay-feasible for chains that already belong to $\mathcal{H}'$, and adds to $\mathcal{H}'$ all the chains placed on those datacenters. The algorithm finishes once there are no further datacenters to visit and no further chains to add, and returns $\mathcal{H}'$.

We first provide some intuition for Alg. 3, and for the validity of the claim. Consider Fig. 3, and assume for simplicity that the network delay is zero, $R = 1$, and $\max_{\mathbf{h}^u \in \mathcal{H}} \hat{C}_u = \max_{s \in \mathcal{S}} C_s = 1$. Hence, once BU places a single chain on a datacenter, the datacenter becomes (almost) full.

Fig. 3 depicts a scenario where BU fails to place chain $\mathbf{h}^{u_6}$. Hence, algorithm 3 assigns $\tilde{u} = u_6$ (ln. 1), and $\mathcal{H}' = \{\mathbf{h}^{u_6}\}$ (ln. 2). Next (ln. 3), the algorithm visits every datacenter on the path from $\overline{s}(u_6) = s_0$ to $p_{u_6} = s_3$. Namely, the algorithm visits s_0, s_1 and s_3, and consequently adds to $\mathcal{H}'$ all the chains placed on these datacenters (ln. 7-8). At this stage, we have $\mathcal{H}' = \{\mathbf{h}^{u_6}, \mathbf{h}^{u_3}, \mathbf{h}^{u_5}, \mathbf{h}^{u_4}\}$.

Next, the algorithm visits the datacenters belonging to $\mathcal{S}_{u_3}$ that were not visited yet, namely, s_2 and s_5. Consequently, the algorithm adds $\mathbf{h}^{u_2}$ and $\mathbf{h}^{u_1}$, that are placed on these datacenters, to $\mathcal{H}'$.

At this stage, after marking each unmarked chain in $\mathcal{H}'$, there are no more un-visited datacenters. The algorithm then halts with $\mathcal{H}' = \{\mathbf{h}^{u_1}, \mathbf{h}^{u_2}, \mathbf{h}^{u_3}, \mathbf{h}^{u_4}, \mathbf{h}^{u_5}, \mathbf{h}^{u_6}\}$. By the definition of the potential placement tree (29), $T_{\mathcal{H}'}$ consists of all the datacenters that are delay-feasible for chains belonging to $\mathcal{H}'$, namely, $T_{\mathcal{H}'} = \{s_0, s_1, s_2, s_3, s_5\}$. Recall that we assume that $R = 1$ and $\max_{u \in \mathcal{U}} \hat{C}_u = C_s = 1$. Hence, $|\mathcal{H}'| = 6 > 5 = \frac{R}{\max_{u \in \mathcal{U}} \hat{C}_u} \sum_{s \in T_{\mathcal{H}'}} C_s$, and the claim holds true.

We now turn to prove the claim. We first show that $\mathcal{T}_{\mathcal{H}'}$ is almost-full. We use induction over the chains added to $\mathcal{H}'$ (ln. 8 in Alg. 3). For the base, we have $\mathcal{H}' = \left\{\mathbf{h}^{\tilde{u}}\right\}$. As BU fails to place $\tilde{u}$, we know that $\mathcal{T}_{\mathcal{H}'} = \mathcal{T}_{\{\mathbf{h}^{\tilde{u}}\}} = \{\mathcal{S}_{\tilde{u}}\}$ is almost-full.

For the induction step, consider a chain $\mathbf{h}^{u'}$ that is added to $\mathcal{H}'$ while considering datacenter s (ln. 8 in Alg. 3). Let $\mathbf{h}^u$ denote the concrete chain considered in ln. 3 of the algorithm at that iteration. Alg. 3 visits datacenters in a top-down order (that is, advancing towards the leaves), while BU visits datacenters in DFS-order, i.e., in bottom-up order (that is, from the leaves towards the root). Furthermore, Alg. 3 handles chains in reverse order of placement by BU. It therefore follows that both $\mathbf{h}^u$ and $\mathbf{h}^{u'}$ were unplaced when BU considered datacenter s, and $s \in \mathcal{S}_u \cap \mathcal{S}_{u'}$. Hence, BU placed $\mathbf{h}^{u'}$ on s before it tried to place $\mathbf{h}^u$. By the induction hypothesis, $\mathcal{S}_u$ is almost-full. Hence, by Lemma 12, $\mathcal{S}_{u'}$ is also almost-full. Hence, after $\mathbf{h}^{u'}$ is added to $\mathcal{H}'$, $\mathcal{T}_{\mathcal{H}'}$ is still almost-full. We therefore proved by induction that when Alg. 3 halts, $\mathcal{T}_{\mathcal{H}'}$ is almost-full.

As $\mathcal{T}_{\mathcal{H}'}$ is almost-full, every datacenter $s \in \mathcal{T}_{\mathcal{H}'}$ is almost-full. Hence, by Lemma 10, BU placed at least $\frac{R \cdot C_s}{\max_{u \in \mathcal{U}} \hat{C}_u}$ chains on each such datacenter s. Hence, the overall number of chains that BU placed on $\mathcal{T}_{\mathcal{H}'}$ is at least $\frac{R}{\max_{u \in \mathcal{U}} \hat{C}_u} \sum_{s \in \mathcal{T}_{\mathcal{H}'}} C_s$. By the construction of $\mathcal{H}'$, *every* chain that BU placed on $\mathcal{T}_{\mathcal{H}'}$ belongs to $\mathcal{H}'$. Hence, $\mathcal{H}'$ contains all the $\frac{R}{\max_{u \in \mathcal{U}} \hat{C}_u} \sum_{s \in T_{\mathcal{H}'}} C_s$ chains that BU placed on $\mathcal{T}_{\mathcal{H}'}$. In addition, $\mathcal{H}'$ includes $\mathbf{h}^{\tilde{u}}$, and the result follows. □

After having characterized the scenarios where BU fails, the following lemma shows a sufficient condition for the problem being infeasible without resource augmentation (recalling the definition of $\tilde{\mu}$ in Eq. (28)).

Lemma 14. *If there exists a set of chains $\mathcal{H}'$ s.t. $|\mathcal{H}'| \cdot \tilde{\mu} > \sum_{s \in T_{\mathcal{H}'}} C_s$, then the problem is infeasible without resource augmentation.*

Proof. Any feasible solution must allocate for each chain $\mathbf{h}^u \in \mathcal{H}'$ at least $\tilde{\mu}$ CPU units on some datacenter(s) in $T_{\mathcal{H}'}$. The result follows. □

The theorem below, which is our main result in this section, now follows from combining Lemma 13 and Lemma 14.

Theorem 15. *Assume BU uses in each datacenter a multiplicative resource augmentation of processing capacity*

$$R = \frac{\max_{u \in \mathcal{U}} \hat{C}_u}{\tilde{\mu}} . \tag{31}$$

Then, BU finds a feasible solution whenever such a solution exists in a system without resource augmentation.

Proof. Assume that BU fails. Assigning the value of R (31) in Lemma 13, there exists a set of chains $\mathcal{H}'$ s.t.

$$|\mathcal{H}'| > \frac{1}{\tilde{\mu}} \sum_{s \in T_{\mathcal{H}'}} C_s.$$

Applying Lemma 14, the result follows. □

VII. Algorithmic solution to the DMP

In this section, we use the algorithms for the CPU allocation problem (Sec. V) and for the placement problem (Sec. VI) as building blocks for solving the DMP. We first present an algorithm that takes as input a feasible solution for DMP, and greedily reduces its cost (Sec. VII-A). Later, we present our algorithmic solution for DMP, BUPU, which targets a minimal cost solution, using the minimum amount of resource augmentation (Sec. VII-B).

A. Reducing costs

We first note that the placement costs and latency constraints are *separable* between different chains. That is, for any two chains $\mathbf{h}^u$, $\mathbf{h}^{u'}$, if $\mathbf{h}^u$ is placed on datacenter s using CPU allocation $\boldsymbol{\mu}^{u,s}$, the cost of the placement and CPU allocation

Algorithm 4 PU($\mathcal{G}, \tilde{\mathcal{H}}, \{\boldsymbol{\mu}^{u,s}\}_{u,s} \{\mathcal{S}_u\}_u$, feasible $\mathbf{y}, \mathbf{a}$)

1: **while** True **do**
2: **for** $\mathbf{h}^u \in \tilde{\mathcal{H}}$ in non-decreasing order of $\mathbf{y}(u,s) \cdot \boldsymbol{\mu}^{u,s}$ **do**
3: push $\mathbf{h}^u$ as high as possible in the tree as long as this decreases $\mathbf{h}^u$'s cost, and update $\mathbf{y}$, $\mathbf{a}$ accordingly
4: **if** did not succeed to push up any chain **then**
5: **break**
6: return $\mathbf{y}, \mathbf{a}$

Algorithm 5 BUPU ($\mathcal{G}, \mathcal{H}, \tilde{\mathcal{H}}, \mathbf{a}$)

1: $\{\boldsymbol{\mu}^{u,s}\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}, s \in \mathcal{S}}, \{\mathcal{S}_u\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}}$ = GFA($\mathcal{G}, \tilde{\mathcal{H}}$)
2: $\mathbf{y}, \mathbf{a}$ = BU ($\mathcal{G}, \tilde{\mathcal{H}}, \{\boldsymbol{\mu}^{u,s}\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}, s \in \mathcal{S}}, \{\mathcal{S}_u\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}}, \mathbf{a}$)
3: **if** $\mathbf{y} \not\equiv 0$ **then** ▷ found a feasible solution
4: $\mathbf{y}, \mathbf{a}$ = PU($\mathcal{G}, \tilde{\mathcal{H}}, \{\boldsymbol{\mu}^{u,s}\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}, s \in \mathcal{S}}, \{\mathcal{S}_u\}_{\mathbf{h}^u \in \tilde{\mathcal{H}}}, \mathbf{y}, \mathbf{a}$)
5: **else**
6: $\mathbf{y}, \mathbf{a}$ = feasible solution with minimal resource augmentation ▷ binary search using BU($\mathcal{G}, \mathcal{H}, \{\boldsymbol{\mu}^{u,s}\}_{u \in \mathcal{U}, s \in \mathcal{S}}, \{\mathcal{S}_u\}_{u \in \mathcal{U}}, \mathbf{a}$)
7: $\mathbf{y}, \mathbf{a}$ = PU($\mathcal{G}, \mathcal{H}, \{\boldsymbol{\mu}^{u,s}\}_{u \in \mathcal{U}, s \in \mathcal{S}}, \{\mathcal{S}_u\}_{u \in \mathcal{U}}, \mathbf{y}, \mathbf{a}$)

of $\mathbf{h}^u$, and the latency inflicted on $\mathbf{h}^u$, are independent of the placement and CPU allocation of $\mathbf{h}^{u'}$.

Based on this observation, we devise our algorithm, *PushUp* (or *PU* for short), for reducing the cost of a given feasible solution. PU, formally described in Alg. 4, scans the provided feasible solution, and greedily tries to improve it by pushing each chain as high up as possible in the network topology (while reducing the cost). PU considers chains in non-increasing order of the CPU units allocated to them, thus prioritizing chains that potentially offer the largest gain from being pushed-up.

Intuitively, this "push-up" operation serves two goals: (i) decreasing the total cost and, (ii) decreasing the total number of migrations, as a datacenter located higher in the tree can serve users located in a larger physical area. Importantly, PU always outputs a feasible solution. Hence, one may run PU as a heuristic to improve a feasible solution found by any algorithm solving the placement problem.

Run-time analysis: PU moves a chain to another datacenter only if this reduces the cost. Hence, once it moves chain $\mathbf{h}^u$ from a datacenter, it never moves it back (due to separability). Thus, the number of iterations of the *while* loop is at most $\left|\tilde{\mathcal{H}}\right| \times \text{height}(\mathcal{G})$. As each iteration requires $O(\left|\tilde{\mathcal{H}}\right| D(\mathcal{G}))$ steps, the time complexity of PU is $O(\left|\tilde{\mathcal{H}}\right|^2 D(\mathcal{G})^2)$.

B. The BUPU algorithm

Theorem 15 provides an upper bound on the amount of resource augmentation that BU requires to find a feasible solution for the placement problem. However, the amount of resource augmentation needed in practice might be significantly lower than that provided by the theorem. Our algorithm for solving the DMP, Bottom-Up Push-Up (BUPU), aims at finding a feasible, minimal-cost solution while using a minimal amount of resource augmentation. The algorithm is summarized in Fig. 4, and formally defined in Alg. 5.

The algorithm gets as input the structure of the network $\mathcal{G}$, and the set of chains $\mathcal{H}$. These inputs are used by all the modules of the algorithm (in Fig. 4 we omit the arrows connecting $\mathcal{G}$ and $\mathcal{H}$ to each module to improve clarity). In addition, BUPU gets the set of critical and newly arriving chains $\mathcal{H}^* \subseteq \mathcal{H}$, and the currently available resources $\mathbf{a}$.

BUPU first runs GFA to obtain for every critical or newly arriving chain $\mathbf{h}^u$ its list of delay-feasible datacenters $\mathcal{S}_u$, and the minimal CPU allocation required for placing $\mathbf{h}^u$ on each server belonging to $\mathcal{S}_u$ (ln. 1). Recall that this minimal CPU allocation is denoted $\boldsymbol{\mu}^{u,s}$. Next, BUPU runs BU to see if a feasible solution exists given the current resource augmentation (ln.2). If so, then PU is applied on the set of critical and newly arriving chains to reduce the cost (ln. 4). However, if BU does not find a feasible solution when considering (re)placing only critical and newly arriving chains, the algorithm turns to solve the placement problem for *all* chains in the system, to ensure feasibility. In this case, BUPU might end up "reshuffling" the placement of many of the chains. To adjust the amount of resource augmentation, BUPU performs a binary search for the minimal amount of resource augmentation required for obtaining a feasible solution (ln. 6). Given a feasible solution that minimizes the amount of resource augmentation, BUPU runs PU on all the chains, to reduce the solution cost (ln. 7).

Intuitively, using more resource augmentation allows locating more chains in the cloud, thus reducing both the computation costs, and the need for future migrations. Hence, there exists a tradeoff between the amount of resource augmentation, and the solution cost. We study this tradeoff in Sec. VIII-C.

VIII. Numerical Evaluation

In this section, we evaluate BUPU against existing alternatives and highlight some trade-offs, thus providing insights that go beyond our analytical results.

A. Simulation settings

We now describe the settings of our baseline scenario, and later vary some of them to study their impact on performance.

Service area. We consider two real-world scenarios, capturing mobility patterns with different characteristics. We focus on the centers of the cities of *Luxembourg* and *Principality of Monaco*, using mobility traces [27], [28] and real-world antenna locations, publicly available in [29]. For each simulated area, we consider the antennas of the cellular telecom provider having the largest number of antennas in the simulated area. The settings of the service areas are detailed in Table II, which details also some traffic parameters, which we shortly explain. For both traces we consider the rush hour period between 7:30 am and 8:30 am.

Network and datacenters. Our simulated networks are illustrated in Fig. 5, where each PoA is co-located with a leaf datacenter. At each decision period, each service chain is associated with the nearest PoA datacenter. Figures 5a and 5d depict a Voronoi diagram of the simulated areas. The network topology connecting the datacenters is a 6-height tree, structured as follows. Denote a topology level by $\ell \in \{0, 1, \ldots, 5\}$, with $\ell = 0$ corresponding to the leaf datacenters (co-located with the PoAs), and $\ell = 5$ corresponding to the root datacenter.

We build levels $5, 4, 3, 2, 1$ in Luxembourg's network by recursively partitioning the simulated area into $1, 4, 16, 64, 256$

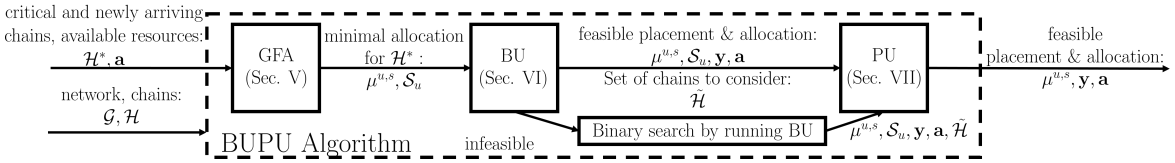


Fig. 4: Structure of the BUPU algorithmic solution.

TABLE II: Simulated scenarios

	simulated area (km^2)	telecom provider	antennas (#PoAs)	#distinct vehicles	avg. traffic (#vehicles/s)	avg. density (#requests/km^2)	linear density (#vehicles/km)	speed (km/h)
Luxemburg	6.8×5.7	Luxembourg Post	1524	25,497	2191	56	2.48	15.4
Monaco	3.1×1	Monaco Telecom	231	13,788	7121	2297	32.7	9.0

rectangles, respectively. Namely, each datacenter at level $\ell \in \{5, 4, 3, 2\}$ has 4 children, each of them responsible for 1/4 of its area. The children of each datacenter at level $\ell = 1$ are the PoAs (antennas and datacenters), of the telecom provider Luxemburg Post, located in its rectangle. Finally, if no PoAs exist in a certain rectangle, the respective datacenters are pruned from the tree. Figures 5b and 5c detail the four top levels in Luxembourg's network.

Monaco's network is built in a similar fashion to that of Luxembourg. However, as Monaco's center makes a long, narrow rectangle (3.1 km $\times$ 1 km), at the top-level (level 5), we partition the simulated area into three horizontal almost-square rectangles. We build levels $3, 2, 1$ by recursively partitioning these squares into quadrants, so that levels $4, 3, 2, 1$ comprise $3, 12, 48$, and 192 datacenters, respectively. Finally, the leaf datacenters are the 231 antennas (and co-located datacenters) of Monaco Telecom. Figures 5e and 5f detail the three highest levels in Monaco's network. Note the pruned datacenters in Fig. 5f, corresponding to areas where no PoAs exist (e.g., areas in the sea).

The datacenter processing capacity depends upon the level ℓ and is set to $\ell \cdot C_{\text{cpu}}$, for a given C_{cpu}, to reflect the increase of datacenter capacities when moving from the edge to the cloud. The total link delay $\tau(i, j)$ is set to 2 ms for every link (i, j), resulting in a maximum round trip network delay of 20 ms from the PoA (level 0) to the root (level 5). Given the minimal allocations obtained by GFA, we use the ILP relaxation of the problem, as discussed in Sec. IV-B. Using this ILP, through binary search, one can find the minimal C_{cpu} for which there exists a feasible solution for the relaxation. This capacity is denoted by $\hat{C}_{\text{cpu}}$, and serves as the baseline value of the CPU available at every server, where we consider resource augmentation with respect to this value. We note that $\hat{C}_{\text{cpu}}$ serves as a lower bound on the required capacity for such capacity allocation settings.

Traffic and mobility. To characterize the traffic in the simulated scenarios, we consider the *linear vehicle density*, defined as the average number of cars per km of lane[3]. Fig. 6 depicts the linear vehicle density within the "coverage area" of each datacenter at levels $1, 2, 3, 4$. Note the significantly higher values in Monaco's scenario, capturing the heavier traffic in this network.

[3]A *lane* is a uni-directional path on the road; a single road may contain one or more lanes in each direction.

Fig. 7 captures the average number of cars that have moved to another rectangle (or left the simulated area) during the sampling period. This can be seen as the offered migration rate, since a car changing cell may dictate migrating the corresponding service chain. The higher traffic density in Monaco is translated to lower mobility.

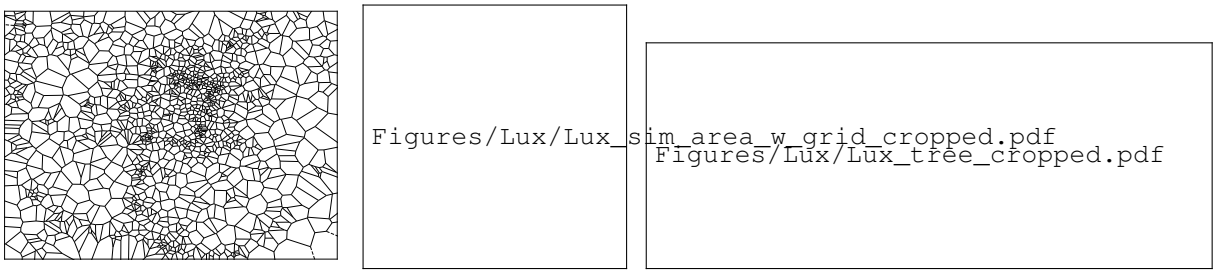
Consider again Table II. The table presents the average number of active vehicles, and the average demand density, defined as the number of service requests (vehicles) per square kilometer. Observe that Monaco's higher linear vehicle density results in a significantly lower average speed (9.0 km/h in Monaco vs. 15.4 km/h in Luxembourg).

Services and service chains. We consider two types of time-critical automotive safety services, one requiring a maximum delay of 10 ms (e.g., collision avoidance [30]) and the other a maximum delay of 100 ms (e.g., see-through [31]). For simplicity of description, we hereinafter refer to the services with the tighter delay constraints of 10 ms as *RT* (real-time) services, and we refer to the corresponding chains as *RT chains*. As reported in [30], a service chain consists of 3 VMs. Also, for each chain $\mathbf{h}^u$, $\theta^u_1\lambda^u_1 = \theta^u_3\lambda^u_3 = 200$ MHz, and $\theta^u_2\lambda^u_2 = 1$ GHz, so as to reflect a chain with a front-end and back-end VM with low computation load and a central VM with high computation load. For simplicity, we assume $\gamma^u_k\theta^u_k$ to be a constant. Upon entering the considered geographical area, each vehicle requests one of the two services at random, with some probability, to be defined later.

Cost parameters. We choose the cost values so that all the three components of our objective function in (11) – namely, computation cost, bandwidth cost, and migration cost – are no more than an order of magnitude apart, and none of them becomes negligible. The cost of 100 MHz of CPU at level ℓ is $\chi^c = 2^{5-\ell}$ cost units, to reflect the decrease in computation costs when moving from the edge to the cloud [**?**], [4]. The bandwidth cost is $\chi^b = 3$ cost units, while the migration cost is $\chi^m = 600$ cost units.

Note that by this choice of parameters, we have the computation cost of a chain greater than $14 \cdot 2^{5-\ell}$ (due to our choice of $\theta \cdot \lambda$ above), which ranges between 14 and 448, depending on the level. Furthermore, the bidirectional bandwidth cost of a chain placed on level ℓ is $3 \cdot 2 \cdot \ell$, which is between 0 and 30, depending on the level.

Benchmark algorithms. We set the default decision period to $T = 1$ s, and assume that there exists a mobility prediction



(a) Luxembourg's cell coverage (b) Luxembourg's datacenters coverage area (c) Luxembourg's datacenters network



(d) Monaco's cell coverage (e) Monaco's datacenters coverage area (f) Monaco's datacenters network

Fig. 5: The service network in Luxembourg (top) and Monaco (bottom). Figures 5a and 5d present Voronoi diagrams of the PoAs, corresponding to the leaves (level 0). Figures 5b and 5e illustrate the iterative partition of the area into rectangles. The rectangles highlighted in yellow, pink, blue, and green, correspond to levels 5 (root), 4, 3, and 2, respectively, in the network, as depicted in Figures 5c and 5f. Some of the leaves in Monaco's datacenters network are pruned from the tree, as no PoAs exist in the respective rectangles.

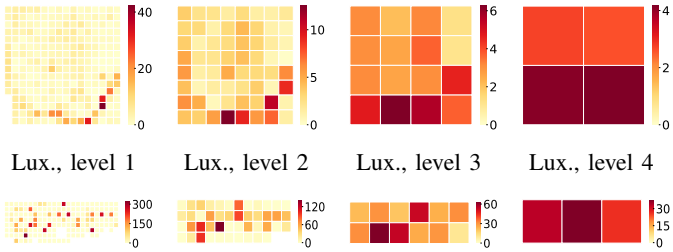


Fig. 6: Average linear vehicular density [vehicles/km] during the 7:30-8:30 am interval in each rectangle in Luxembourg (top) and Monaco (bottom). White rectangles correspond to areas where no roads exist.

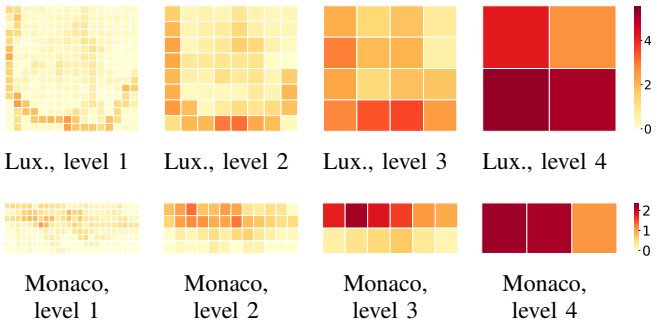


Fig. 7: Average number of vehicles leaving a rectangle every second during the 7:30-8:30 am interval in Luxembourg (top) and Monaco (bottom).

scheme that associates each vehicle with the closest PoA in the next decision period. Based on this prediction, we run GFA to get the CPU allocation $\boldsymbol{\mu}$ for each chain, as detailed in Sec. V. This allows us to fix the CPU allocations and compare our approach to existing solutions for the placement problem. Notably, if running placement without GFA, alternative deployment solutions (not dealing with CPU allocation and unaware of all the constraints in the DMP) will get infeasible solutions most of the time. For our comparison, we consider the following benchmarks:

Lower-bound (LBound): Given the allocation, we use the ILP discussed in Sec. IV-B and solve the linear relaxation of this problem. This provides a lower bound on the cost of any feasible solution for the placement problem. In contrast to our algorithm BUPU, the fractional solution may place parts of the same chain (or even "fractions of VMs") on distinct datacenters. Furthermore, the LP formulation considers at each iteration all the chains in the system (not only the critical or newly arriving chains), thus significantly increasing the possible solution space. Hence, LBound provides a lower bound on the minimal cost for the placement problem.

First-fit (F-Fit): This scheme places each chain on the first delay-feasible datacenter with sufficient available capacity on the path from the root to the chain request's PoA.

CPVNF [8]: This algorithm orders the critical and newly-arriving chains in a non-increasing order of the CPU capacity they require, if placed on an edge datacenter. It then places each such chain on the feasible available datacenter incurring the lowest cost according to (11). This benchmark is an adaptation to our problem of the CPVNF algorithm [8], which was used as a benchmark also in [32].

Feasibility. Our BUPU algorithm, as well as F-Fit and CPVNF, first considers only critical and newly arriving chains; if merely (re)placing these chains does not yield a feasible

solution, the algorithm considers placing from scratch *all* the chains in the system. Thus, referring to the framework presented in Fig. 4, the benchmark algorithms (F-Fit and CPVNF) are used instead of BU and PU. Note that our comparison methodology over-estimates the performance of our benchmark algorithms, as we give them the same (optimal) solution for the CPU allocation problem (found by GFA) "for free".

Simulation methodology. The performance of LBound is deterministic. However, the performance of F-Fit, CPVNF and BUPU depends upon the arbitrary order of handling requests to which the algorithm in question gives the same priority. Hence, in each experiment we run each of these three algorithms 20 times, considering a random order of handling the requests.

The simulator has been developed in Python and it is fully available on [33].

B. Resources required for finding a feasible solution

As the first step, we study the resource augmentation required by each algorithm to find a feasible solution. We focus on ten minutes of the trace, referring to a busy morning rush hour (08:20-08:30), with 6,859 and 9,351 distinct vehicles passing in the simulated area in Luxembourg and Monaco, respectively. We vary the ratio of RT chains, i.e., corresponding to requests with a maximum delay of 10 ms. For each setting, we use binary search to find the minimum amount of resources (captured by the minimum CPU at the leaf datacenters) required by the considered algorithm to find a feasible solution for *every* 1-second slot along the 10-minute trace.

Fig. 8 shows the results of this experiment. The amount of processing capacity required by BUPU is extremely close to the lower bound (LBound) in the Luxembourg scenario (Fig. 8a), and perfectly matches it in the Monaco one (Fig. 8b). In contrast, the processing capacity required by CPVNF and F-Fit for finding a feasible solution is much higher: in Luxembourg's scenario, CPVNF and F-Fit typically need a processing capacity that is 50%-100% higher than the capacity required by BUPU.

As expected, the amount of resources required for obtaining a feasible solution consistently increases for a larger fraction of RT chains. This happens because tighter timing constraints may require allocating more CPU resources for each chain, to decrease the computational delay. Further, RT service requirements also dictate placing the chain close to the edge datacenter, thus reducing the use of processing capacities at higher-level datacenters. However, in the Luxembourg scenario, the processing capacity required by BUPU with 100% of RT chains is still lower than that required by CPVNF and F-Fit when no RT chain is present. Finally, comparing the Luxembourg and Monaco scenarios, we note that the higher car density in Monaco, and the smaller number of leaf-datacenters in that scenario (only 231 in Monaco, compared to 1,524 in Luxembourg), dictate using higher computation capacity in the leaf datacenters for finding a feasible solution.

TABLE III: Normalized cost of chain deployment and migration vs. resource augmentation. The costs are normalized with respect to the cost obtained by the lower bound (LBound). An infinite cost indicates that the algorithm cannot find a feasible solution

Luxembourg			
$C_{cpu}/\hat{C}_{cpu}$	Normalized Cost		
	BUPU	F-Fit	CPVNF
1.00	∞	∞	∞
1.06	1.76	∞	∞
1.50	1.91	∞	∞
2.00	1.17	∞	∞
2.35	1.06	1.06	∞
2.40	1.06	1.06	1.06
2.50	1.05	1.05	1.05

Monaco			
$C_{cpu}/\hat{C}_{cpu}$	Normalized Cost		
	BUPU	F-Fit	CPVNF
1.000	∞	∞	∞
1.002	1.36	∞	∞
1.50	1.08	∞	∞
1.58	1.09	1.10	1.10
2.00	1.05	1.05	1.05
2.50	1.01	1.01	1.01

TABLE IV: Normalized migration cost vs. resource augmentation

	$C_{cpu}/\hat{C}_{cpu}$		
	1.10	1.50	2.00
Luxembourg	639,609	556,739	84,138
Monaco	194,236	12,032	8,702

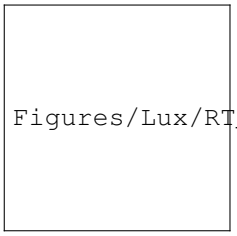
C. Cost comparison

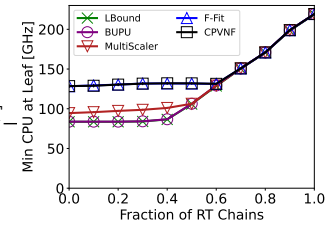
We now consider 30% of RT chains and compare the costs obtained by different algorithms. Similarly to Sec. VIII-B, here we also focus on ten minutes of the trace, referring to a busy morning rush hour (08:20-08:30).

Tab. III shows the average cost per second for each algorithm when varying the resource augmentation factor, $C_{\text{cpu}}/\hat{C}_{\text{cpu}}$. To make a meaningful comparison, the costs are normalized with respect to the cost obtained by LBound for the same amount of resources. The table shows the normalized costs when the resource augmentation factor varies between 1 and 2.5. In addition, for each scenario and algorithm, the table presents the minimal amount of resource augmentation for which the considered algorithm finds a feasible solution with a confidence level of at least 99%. When no feasible solution is found, the corresponding cost is infinite.

Interestingly, for a wide range of resource augmentation, *only* BUPU finds a feasible solution. When the resource augmentation is very small (e.g., only 100.2% of the resources required by LBound for finding a feasible solution in the Monaco scenario), the cost of BUPU's solution is significantly higher than that of LBound. However, when increasing the amount of resources, the gap between BUPU and LBound reduces very substantially. As for F-Fit and CPVNF they need a significant resource augmentation for finding a feasible solution, e.g., 2.4 times the processing capacity used by LBound in the Luxembourg scenario. Finally, for a high amount of resource augmentation (i.e., when resources are abundant), all strategies provide a solution of comparable cost.

Tab. IV, instead, captures the impact of the resource aug-





(a) Luxembourg (b) Monaco

Fig. 8: Minimum required processing capacity for finding a feasible solution when varying the ratio of RT service requests.

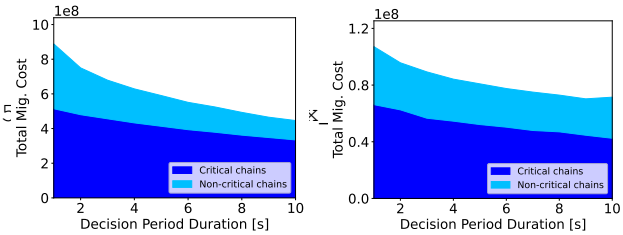
mentation on the *migration cost* experienced by BUPU: the larger the resource augmentation, the lower the migration cost is. Indeed, a tight resource budget forces BUPU to migrate many, even non-critical, chains to find a feasible solution. On the other hand, high resource augmentation allows BUPU to find a feasible solution while placing more chains at a higher network level, thus reducing the current overall cost and mitigating the need for future migrations when the users move.

D. Decision period

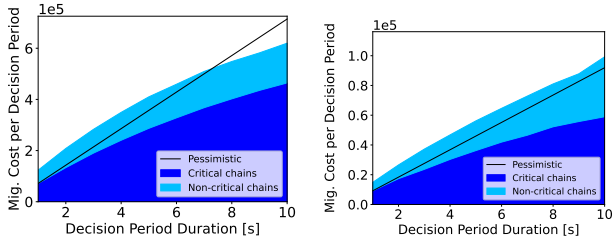
Next, we focus on the impact of decision period T on the migration cost and on the SLA violation. To this end, we consider the 07:30-08:30 trace, and vary T. The amount of CPU is set to 110% the minimal capacity that BUPU needs for finding a feasible solution when $T = 1$. This choice enforces tight capacity constraints, while being sufficient for allowing BUPU to find a feasible solution, even when varying decision period T.

Fig. 9a depicts the cost of migrating critical and non-critical chains, as T varies. The overall migration cost is governed by *non-compulsory migrations* (namely, migrations of non-critical chains) that the algorithm occasionally performs, as it cannot find a feasible solution otherwise. In such a case the algorithm is forced to "reshuffle" the placement of possibly many chains, incurring a high overall migration cost (well beyond that imposed by critical chains alone). Clearly, increasing T leads to a lower cost of such non-compulsory migrations, as a smaller T implies reducing the number of times the algorithm runs during the 1-hour trace, and in particular it reduces the number of such potential "reshuffles". The migration of critical chains, instead, may be considered *compulsory*, and is determined mainly by the user mobility, and not by decision period T. Correspondingly, in the Monaco scenario, the cost of migrating critical chains hardly changes when varying T. In the Luxembourg scenario, however, the cost of critical chains migration slightly decreases when increasing T.

To better understand this phenomenon, consider Fig. 9b, showing the cost of migrating critical and non-critical chains, but now normalized per decision (i.e., per single run of BUPU). We consider that every second, some number X of chains become critical. One can consider a *pessimistic* estimator, such that when using a decision period of T seconds, the number of chains that become critical during the decision period is $X \cdot T$. In the Monaco scenario, the per-decision



(a) Cost of migrating critical and non-critical chains during the 1-hour trace in Luxembourg (left) and in Monaco (right).



(b) The per-decision cost of migrating critical and non-critical chains in Luxembourg (left) and Monaco (right). The black line depicts our pessimistic estimator.

Fig. 9: Impact of T on migration cost.

migration cost is indeed very close to our pessimistic estimator. This phenomenon can be attributed to the low mobility of vehicles in this trace, captured by the low average speed (recall Tab. II). However, when vehicles move faster, as in the Luxembourg scenario, a sufficiently large T may translate to a single migration (per vehicle, per decision period), as opposed to several migrations of that vehicle when using a decision period of one second. This translates to a lower migration cost, compared to our pessimistic estimator. However, this comes at the cost of SLA violation.

To study such violations incurred by increasing the decision period T, we considered the average violation time of critical chains. As expected, with a decision period T an average SLA violation lasting roughly $T/2$ will be experienced. This is consistent with a model where each critical chain starts experiencing an SLA violation at a random time instant, uniformly distributed in $(0, T]$. We verified this behaviour in our evaluation for both the Monaco and Luxembourg scenarios.

Finally, we note that at each run, there is some probability that BUPU also migrates non-critical chains for finding a feasible solution, and this probability need not depend upon T. This indeed follows by considering the light-blue zone which is an almost constant additive increase of the migration cost, on top of that related to critical chains.

IX. Related work

Service migration has been extensively investigated in recent years. The works [34], [35] address a dynamic VNF placement, where chains are migrated to serve mobile users. These studies disregard the computational delay, focusing on network delay solely. [34] designs efficient algorithms with strong performance guarantees – e.g., $O(1)$ approximation ratio for the network cost, (i.e., the average user-datacenter distance), while using an $O(1)$ resource augmentation on the

datacenter's capacities. However, [34] interprets the network delay as a part of the objective function, and not as a constraint. Furthermore, [34] assumes that all current and future users' locations are known in advance.

More realistic assumptions on the knowledge of users' locations are made in [1], [36], where the migration problem is modeled as a Markov decision process. Nonetheless, such an analysis relies on some assumptions about users' mobility (e.g., a stationary system with known, or predictable, transition probabilities), which do not conform with realistic scenarios.

Highly heterogeneous and unpredictable user mobility is considered in [2], [8], [11], [12], which envision algorithms that consider various optimization criteria. CPVNF [8] is a greedy approach that we use as a benchmark and discuss in Sec. VIII-A. [2] decreases the migration overhead by clustering users, thus shrinking the amount of data migrated between datacenters. However, the model used in [2] assumes that the migration destination always has enough resources, and ignores computational delay. This model may conform with cloud computing where computational resources are abundant, and the primary source of delay is network delay, but not with edge computing, where the scarcity of resources and the tight delay constraint dictate considering the computational delay. [11] considers multiple optimization criteria, i.e., the amount of resources consumed by migration, migration time, and service downtime. However, the model used in [11] allows declining a migration request, or handling a request while breaking its target delay. In contrast, we address the problem of handling all the migration requests, while satisfying target delay constraints. In a fog computing scenario, [12] selects for each migration request a destination, based on the topological distance from the user, the availability of resources in the destination, and the data protection level in the destination. However, [12] focuses on a selfish optimization of the migration destination for a single user, while we target finding a feasible global solution, while minimizing the overall system cost.

Other works formulate the migration problem as an ILP [4], [13], MILP [9], or Mixed-Integer Quadratic Program [15], but none of them guarantees to find a feasible solution (possibly using more resources). In particular, these solutions usually handle requests in parallel (the ILP/MIQP solvers), or in an arbitrary order, e.g., based on the request time [11]. As a result, requests with relaxed delay constraints may be deployed on a scarce resource, possibly making the resource unavailable for tighter-delay applications, that cannot be deployed elsewhere. Our solution, on the contrary, orders requests based on their delay constraints, before handling them.

Some studies address predicting future users' mobility and traffic fluctuation [37]–[39]. However, these works differ from ours by both the objective function, and the tools used. The works in [40], [41] decrease the migration overhead (migration time, service downtime, and quantity of resources consumed by migration) by optimally deciding which data to migrate and in what order and pace. [42] considers multiple simultaneous migration requests, and focuses on scheduling the migrations and determining the bandwidth allocated to each migration process to minimize the total migration time and the service level objective violation. [13] uses the optimal stopping theory to optimize the length of the decision period between subsequent runs of the migration algorithm.

Some of the optimizations mentioned above are orthogonal to our work, and hence could be incorporated into our solution to boost performance. Finally, implementation issues and performance overhead of VM live migration are discussed in [43].

X. Conclusions

We tackled mobile service provisioning in the edge-cloud continuum, envisioning algorithmic solutions with provable guarantees in terms of solution feasibility and resource usage. Besides addressing service chain deployment and resource allocation, our solution fulfills the service delay requirements and tackles service migration by deciding which chains should be migrated and, if migrating, towards which datacenter. Our numerical results, derived in large-scale, vehicular scenarios, highlight interesting trade-offs and show that our approach may provide a feasible solution by using half the quantity of computing resources required by state-of-the-art alternatives. We also show the robustness of the proposed approach, when varying the considered scenarios and the decision period of the algorithm.

References

[1] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 369–382, 2016.

[2] R. Bruschi, F. Davoli, P. Lago, and J. F. Pajo, "Move with me: Scalably keeping virtual objects close to users on the move," in *IEEE ICC*, 2018, pp. 1–6.

[3] B. Kar, K.-M. Shieh, Y.-C. Lai, Y.-D. Lin, and H.-W. Ferng, "QoS violation probability minimization in federating vehicular-fogs with cloud and edge systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, 2021.

[4] D. Zhao, G. Sun, D. Liao, S. Xu, and V. Chang, "Mobile-aware service function chain migration in cloud–fog computing," *Future Generation Computer Systems*, vol. 96, pp. 591–604, 2019.

[5] Y.-D. Lin, C.-C. Wang, C.-Y. Huang, and Y.-C. Lai, "Hierarchical cord for NFV datacenters: resource allocation with cost-latency tradeoff," *IEEE Network*, vol. 32, no. 5, pp. 124–130, 2018.

[6] A. Ullah, H. Dagdeviren, R. C. Ariyattu, J. DesLauriers, T. Kiss, and J. Bowden, "Micado-edge: Towards an application-level orchestrator for the cloud-to-edge computing continuum," *Journal of Grid Computing*, vol. 19, no. 4, pp. 1–28, 2021.

[7] S. Svorobej, M. Bendechache, F. Griesinger, and J. Domaschka, "Orchestration from the cloud to the edge," *The Cloud-to-Thing Continuum*, pp. 61–77, 2020.

[8] M. Dieye, S. Ahvar, J. Sahoo, E. Ahvar, R. Glitho, H. Elbiaze, and N. Crespi, "CPVNF: Cost-efficient proactive VNF placement and chaining for value-added services in content delivery networks," *IEEE Transactions on Network and Service Management*, pp. 774–786, 2018.

[9] H. Hawilo, M. Jammal, and A. Shami, "Orchestrating network function virtualization platform: Migration or re-instantiation?" in *IEEE CloudNet*, 2017, pp. 1–6.

[10] I. Cohen, G. Einziger, M. Goldstein, Y. Sa'Ar, G. Scalosub, and E. Waisbard, "Parallel VM deployment with provable guarantees," in *IFIP Networking*, 2021, pp. 1–9.

[11] G. Sun, D. Liao, D. Zhao, Z. Xu, and H. Yu, "Live migration for multiple correlated virtual machines in cloud-based data centers," *IEEE Transactions on Services Computing*, pp. 279–291, 2015.

[12] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, and G. Merlino, "Companion fog computing: Supporting things mobility through container migration at the edge," in *IEEE International Conference on Smart Computing (SMARTCOMP)*, 2018, pp. 97–105.

[13] I. Leyva-Pupo, C. Cervelló-Pastor, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic scheduling and optimal reconfiguration of UPF placement in 5G networks," in *ACM MSWiM*, 2020, pp. 103–111.

[14] T. Mahboob, Y. R. Jung, and M. Y. Chung, "Dynamic VNF placement to manage user traffic flow in software-defined wireless networks," *Journal of Network and Systems Management, Springer*, pp. 1–21, 2020.

[15] X. Sun and N. Ansari, "PRIMAL: Profit maximization avatar placement for mobile edge computing," in *IEEE ICC*, 2016, pp. 1–6.

[16] J. Martín-Pérez, F. Malandrino, C.-F. Chiasserini, and C. J. Bernardos, "OKpi: All-KPI network slicing through efficient resource allocation," in *IEEE INFOCOM*, 2020, pp. 804–813.

[17] "Ieee 802.1ax-2008 standard," 2008.

[18] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IEEE IFIP Networking*, 2015, pp. 1–9.

[19] T. Sato and E. Oki, "Program file placement strategies for machine-to-machine service network platform in dynamic scenario," *IEICE Transactions on Communications*, 2020.

[20] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint VNF placement and CPU allocation in 5G," in *IEEE INFOCOM*, 2018, pp. 1943–1951.

[21] F. B. Jemaa, G. Pujolle, and M. Pariente, "QoS-aware VNF placement optimization in edge-central carrier cloud architecture," in *IEEE GLOBECOM*, 2016, pp. 1–7.

[22] R. Gouareb, V. Friderikos, and A.-H. Aghvami, "Virtual network functions routing and placement for edge cloud latency minimization," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, 2018.

[23] B. Kar, E. H.-K. Wu, and Y.-D. Lin, "Communication and computing cost optimization of meshed hierarchical NFV datacenters," *IEEE Access*, vol. 8, pp. 94 795–94 809, 2020.

[24] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the Internet*. Springer Science & Business Media, 2001.

[25] M. Nguyen, M. Dolati, and M. Ghaderi, "Deadline-aware SFC orchestration under demand uncertainty," *IEEE Transactions on Network and Service Management*, pp. 2275–2290, 2020.

[26] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[27] L. Codecá, R. Frank, S. Faye, and T. Engel, "Luxembourg SUMO traffic (LuST) scenario: Traffic demand evaluation," *IEEE Intelligent Transportation Systems Magazine*, pp. 52–63, 2017.

[28] L. Codeca and J. Härri, "Monaco SUMO traffic (MoST) scenario: A 3D mobility scenario for cooperative ITS," *EPiC Series in Engineering*, vol. 2, pp. 43–55, 2018.

[29] "Opencellid," https://opencellid.org/, accessed on 3.10.2021.

[30] G. Avino, P. Bande, P. A. Frangoudis, C. Vitale, C. Casetti, C. F. Chiasserini, K. Gebru, A. Ksentini, and G. Zennaro, "A MEC-based extended virtual sensing for automotive services," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1450–1463, 2019.

[31] F. Rameau, H. Ha, K. Joo, J. Choi, K. Park, and I. S. Kweon, "A real-time augmented reality system to see-through cars," *IEEE Transactions on Visualization and Computer Graphics*, pp. 2395–2404, 2016.

[32] T. Gao, X. Li, Y. Wu, W. Zou, S. Huang, M. Tornatore, and B. Mukherjee, "Cost-efficient VNF placement and scheduling in public cloud networks," *IEEE Transactions on Communications*, pp. 4946–4959, 2020.

[33] "Service function chains migration." [Online]. Available: https://github.com/ofanan/SFC_migration

[34] Y. Fairstein, S. J. Naor, and D. Raz, "Algorithms for dynamic NFV workload," in *International Workshop on Approximation and Online Algorithms*. Springer, 2018, pp. 238–258.

[35] D. Eisenstat, C. Mathieu, and N. Schabanel, "Facility location in evolving metrics," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2014, pp. 459–470.

[36] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, pp. 712–725, 2018.

[37] L. Tang, X. He, P. Zhao, G. Zhao, Y. Zhou, and Q. Chen, "Virtual network function migration based on dynamic resource requirements prediction," *IEEE Access*, vol. 7, pp. 112 348–112 362, 2019.

[38] A. Al-Dulaimy, J. Taheri, A. Kassler, M. R. H. Farahabady, S. Deng, and A. Zomaya, "MULTISCALER: A multi-loop auto-scaling approach for cloud-based applications," *IEEE Transactions on Cloud Computing*, 2020.

[39] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM*, 2019, pp. 1468–1476.

[40] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile VM handoff for edge computing," in *ACM/IEEE SEC*, 2017, pp. 1–14.

[41] R. Stoyanov and M. J. Kollingbaum, "Efficient live migration of linux containers," in *ISC High Performance*. Springer, 2018, pp. 184–193.

[42] T. He, A. N. Toosi, and R. Buyya, "SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds," *Journal of Systems and Software*, vol. 176, p. 110943, 2021.

[43] S. Ramanathan, K. Kondepu, M. Razo, M. Tacca, L. Valcarenghi, and A. Fumagalli, "Live migration of virtual machine and container based mobile core network components: A comprehensive study," *IEEE Access*, vol. 9, pp. 105 082–105 100, 2021.