A comparison of static and dynamic micro-service placement strategies for edge computing

Davide Agostini
University of Modena and Reggio Emilia
Modena, Italy
283804@studenti.unimore.it

Thiago Alves de Queiroz
Institute of Mathematics and Technology
Federal University of Catalão
Catalão-GO, Brazil
taq@ufcat.edu.br

Manuel Iori
Dept. of Science and Methods for Engineering
University of Modena and Reggio Emilia
Reggio Emilia, Italy
manuel.iori@unimore.it

Riccardo Lancellotti
Dept. of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia
Modena, Italy
riccardo.lancellotti@unimore.it

***Abstract*—The advent of edge computing leverages large, distributed infrastructures of edge nodes for executing complex applications. These applications are typically based on the composition of multiple micro-services. The problem of optimal allocation of micro-services to the infrastructure is a typical problem that aims to reduce power consumption and response time. However, as the workload changes, a trade-off arises between the need to define new deployments to support the changed system conditions and avoiding complex re-configurations of the infrastructure that may negatively impact the system availability (due to the need to switch on and off nodes and due to the number micro-services migrations).**

We propose an optimization model based on dynamic programming that is explicitly designed to reduce the disruption on the edge node infrastructure by limiting the number of reconfiguration operations while still guaranteeing the respect of Service Level Agreements in terms of average response time for the various applications. Our model considers the co-existence of micro-services belonging to different applications on the same nodes as well as the impact of network delays. A comparison with a naive alternative that defines a new deployment every time that the workload changes demonstrates that the proposed dynamic programming approach can reduce the number of infrastructure re-configurations by at least 75% and the number of nodes switched on/off by 72%, while still guaranteeing that the power consumption is close to the minimum and response time can still satisfy the SLA requirements.

I. Introduction

Edge computing is a novel paradigm for the execution of complex applications where the computation is pushed as close as possible to the users or to the data sources. Mobile applications for end users can benefit from the reduction in network latency compared to a cloud-based alternative, while IoT applications can benefit from the possibility to filter, aggregate, and pre-process data close to the sensors, thus reducing the amount of data to transfer to the cloud data centers.

A typical application designed for edge computing is composed of several micro-services following a mash-up approach. These micro-services are sequentially executed in such a way that the output of a micro-service is the input of the subsequent one [1]. Orchestrators can either rely on simple strategies for deployment (e.g., placement with overload avoidance constraints [2]) or can aim to define an optimization problem (either to reduce response time [1] or to minimize power consumption and/or costs by powering off some nodes [3]).

However, once the deployment has been implemented, changes in the incoming request flows may determine the need to continuously adapt the micro-service deployment over the edge infrastructure. In some cases, this adaption requires a new execution of the optimization algorithm [3], while other approaches involve a deployment planning that spans multiple re-deployment slots in the future [4]. As most micro-services rely on containerized infrastructures that do not support live migration like VMs, re-deployment leads to some level of service disruption. In this paper, we follow a slightly different approach, formulating a new optimization problem aiming to minimize the amount of re-configuration to satisfy the Service Level Agreement (SLA) requirements while maintaining the power consumption to be as low as possible. In particular, the main contribution of our study can be summarized as follows:

- We define a model based on dynamic programming to define how micro-services should be migrated and how to reconfigure the infrastructure as the workload changes over time;
- We define two different operating scenarios for the testing of such problems;
- We test our proposal against a traditional static programming approach, demonstrating our solution's advantages.

The rest of the paper is organized as follows. Section II presents the problem of micro-service placement in edge computing outlining the performance and energy models as well as the static and the dynamic programming models used to solve this problem. Section III discusses the experiments carried out

to compare the traditional model based on static programming and our proposal based on dynamic programming. Finally, Section IV provides some concluding remarks and outlines the future research directions.

II. Placement strategies

A. Problem overview

We now formally model the problem that is at the core of our research.

We assume to have a set $\mathcal{A}$ of applications that are composed of a sequence of micro-services. Let $a \in \mathcal{A}$ be a generic application. Application a is composed of a sequence of micro-services $m \in \mathcal{M}_a$ that must be executed. It receives at time t requests with a rate $\lambda_a(t)$, and the user-perceived response time for the application is $R_a(t)$. We consider that an SLA for each application a requires the average response time to be: $R_a(t) \leq T_a^{SLA}$, for every considered time t.

Let $\mathcal{M} = \bigcup_{a \in \mathcal{A}} \mathcal{M}_a$ be the set of all micro-services. Each micro-service $m \in \mathcal{M}$ composing an application is characterized by an average service time S_m and a standard deviation σ_m in the execution time.

The micro-services composing the applications are deployed over an infrastructure of edge nodes $e \in \mathcal{E}$. The computational capacity of each edge node determines a speedup C_e such that the average service time of micro-service m is S_m/C_e when the service is executed on edge node e.

In this paper, we consider that power consumption for an edge node can be modeled using a linear formulation, where the power consumption ranges from P_e^0, that is, the power consumption when the edge node e is idle, to P_e^M when the edge node is fully utilized (as described in Sec. II-C).

We define the micro-service placement problem as a multi-objective optimization problem with two separate goals:

- Minimize power consumption of the infrastructure;
- Minimize average execution for each application modeled as a chain of micro-services.

In our problem, we consider time as a discrete sequence of time slots indexed by t. Placement of a micro-service over an edge node at time t is represented using a binary variable $x_{m,e}(t)$, where $x_{m,e}(t) = 1$ means that micro-service m is hosted on node e at time t, while $x_{m,e}(t) = 0$ means that the micro-service is hosted elsewhere. Edge nodes can be turned on or off. This is represented by a second binary variable $y_e(t)$, taking the value 1 if the edge node e is turned on and 0 otherwise. Edge nodes that are turned off will not host any micro-service, that is $x_{m,e}(t) = 0, \forall m \in \mathcal{M}$ and $\forall e : y_e(t) = 0$.

Throughout the paper, we refer to Table I as a summary of the notation adopted in this paper.

TABLE I: Notation and parameters for the proposed model.

Model parameters	
$\mathcal{M}$	Set of micro-services
$\mathcal{M}_a$	Set of micro-services composing application a
$\mathcal{E}$	Set of edge nodes
$\mathcal{A}$	Set of applications
$\lambda_m(t)$	Request rate to micro-service m at time t
$\lambda_e(t)$	Request rate to edge node e at time t
$\lambda_a(t)$	Request rate to application a at time t
$\Lambda(t)$	Global request rate at time t
S_m	Average service time for micro-service m
σ_m	Standard deviation of S_m
C_e	Computational power of edge node e
$S_e(t)$	Average service time on edge node e at time t
$\sigma_e(t)$	Standard deviation of S_e at time t
$W_e(t)$	Average waiting time on edge node e at time t
$R_a(t)$	Average response time for application a at time t
T_a^{SLA}	SLA requirement of application a
$o_{m,n}$	execution order of micro-services m, n
$\delta_{e,f}(t)$	network delay between edge nodes e and f
P_e^0	Power consumption of edge node e when idle
P_e^M	Maximum power consumption of edge node e
Model indices	
e	An edge node
a	An application
m	A micro-service
t	A time slot
Decision variables	
$x_{m,e}(t)$	Allocation of micro-service m to edge e at time t
$y_e(t)$	Use of edge node e at time t

B. Performance model

Let us now focus on a single edge node turned on at time t. The performance of such an edge node can be modeled according to queuing theory as in [1]. The service processing time at node e is an aleatory variable whose probability density follows a mixture of distributions related to the hosted micro-services. The mixture has an average service $S_e(t)$ and a variance $\sigma_e^2(t)$ defined as:

$$S_e(t) = \frac{1}{C_e} \cdot \sum_{m \in \mathcal{M}} x_{m,e}(t) \frac{\lambda_m(t)}{\lambda_e(t)} S_m \tag{1}$$

$$\sigma_e^2(t) = \left(\frac{1}{C_e^2} \cdot \sum_{m \in \mathcal{M}} x_{m,e}(t) \frac{\lambda_m(t)}{\lambda_e(t)} (S_m^2 + \sigma_m^2) \right) - S_e^2(t) \tag{2}$$

In Eqs. (1) and (2), $\lambda_e(t)$ is the arrival request rate on edge node e at time t defined as $\lambda_e(t) = \sum_{m \in \mathcal{M}} x_{m,e}(t) \lambda_m(t)$.

Under the assumption that the arrival process follows a Poisson distribution, we can apply the Pollaczek-Khinchin formula to describe the waiting time $W_e(t)$ of an M/G/1 system:

$$W_e(t) = \frac{S_e^2(t) + \sigma_e^2(t)}{2} \cdot \frac{\lambda_e(t)}{1 - \lambda_e(t) S_e(t)} \tag{3}$$

Previous studies [1] demonstrated that this model is accurate as long as either several micro-services are located on the same edge node or the coefficient of variation (that is, the ratio between standard deviation σ_m and mean value S_m) of the service time of the micro-services is close to 1.

Based on this result, we can describe the response time of an application a as the sum of: (i) the service times of the composing micro-services, (ii) the waiting time on the edge nodes hosting the micro-services, and (iii) the network delays. We can formalize this result as:

$$R_a(t) = \sum_{\substack{m \in \mathcal{M}_a \\ e \in \mathcal{E}}} x_{m,e}(t) \left(W_e(t) + \frac{S_m}{C_e} \right) + \\ + \sum_{\substack{m,n \in \mathcal{M}_a \\ e,f \in \mathcal{E}}} o_{m,n} x_{m,e}(t) x_{n,f}(t) \delta_{e,f}(t) \quad (4)$$

In Eq. (4), $o_{m,n}$ is a parameter used to identify consecutive micro-services in a service chain: $o_{m,n} = 1 \iff m \lessdot n$, meaning that service m is invoked just before n in the service chain.

C. Energy model

We assume that power consumption $P_e(t)$ at edge node e at time t depends only on the node utilization, with a linear function correlating node utilization and power consumption as in [5].

$$P_e(t) = y_e(t) P_e^0 + (P_e^M - P_e^0) \lambda_e(t) S_e(t) \quad (5)$$

The energy consumption of edge node e depends on two factors: a minimum power if the node is turned on (i.e, $y_e(t) = 1$), and a contribution proportional to the node utilization (that is $\lambda_e(t) S_e(t)$).

It is worth noting that, aiming to minimize power consumption and assuming that the edge infrastructure is composed of homogeneous nodes, the problem can be reduced to minimize the number of turned-on edge nodes, that is $\sum y_e(t)$.

D. Static placement problem

Given the previously described model for the energy and for the performance of an edge infrastructure, we can define the problem of the static placement of micro-services over the infrastructure as follows:

$$\min P(t) = \sum_{e \in \mathcal{E}} P_e(t) \quad (6)$$

$$\min R(t) = \sum_{a \in \mathcal{A}} \frac{\lambda_a(t)}{\Lambda(t)} R_a(t) \quad (7)$$

subject to:

$$\sum_{e \in \mathcal{E}} x_{m,e}(t) = 1, \quad \forall m \in \mathcal{M}, \quad (8)$$

$$\lambda_e(t) S_e(t) \leq (1 - \epsilon) y_e(t), \quad \forall e \in \mathcal{E}, \quad (9)$$

$$R_a(t) \leq T_a^{SLA}, \quad \forall a \in \mathcal{A}, \quad (10)$$

$$x_{m,e}(t) \in \{0, 1\}, \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \quad (11)$$

$$y_e(t) \in \{0, 1\}, \quad \forall e \in \mathcal{E}, \quad (12)$$

The variables describing the state of the infrastructure that is, $x_{m,e}(t)$ for micro-service placement and $y_e(t)$ for the power state of the edge node, are the decision variables of the problem.

Eq. (6) is the first objective function of the problem and is related to minimizing the energy consumption of the infrastructure. The power consumption $P_e(t)$ is defined in Eq. (5). The second objective function, to be optimized as long as the power consumption remains minimal, is described in Eq. (7) and refers to the average response time of the applications (weighted according to their invocation frequency). The response time $R_a(t)$ is based on the performance model in Eq. (4). The two objective functions can be combined into a single objective $Obj(t) = P(t) + \varepsilon \cdot R(t)$, where ε is a small value and means that the second goal is taken into account as long as the energy consumption remains minimum.

The solution space of the optimization problem is bounded by these constraints:

- Eq. (8) forces the allocation of each micro-service to exactly one edge node;
- Constraint (9) avoids overload condition for every edge node ($\lambda_e(t) S_e(t)$ is the load on edge node e at time t). They mean that the load must remain strictly below 1 thanks to the arbitrarily small value ϵ, but for turned-off nodes (that is when $y_e(t) = 0$), the load must be 0;
- Constraint (10) captures the SLA requirement of each service chain. In accordance with the literature [4], we consider $T_a^{SLA} = K \cdot \sum_{m \in \mathcal{M}_a} S_m$, where $K = 10$;
- Eqs. (11) and (12) describe the Boolean nature of the decision variables $x_{m,e}(t)$ and $y_e(t)$.

E. Dynamic placement problem

The dynamic placement problem assumes to have a deployment already available at time $t - 1$ that is described by parameter $\bar{x}_{m,e}(t-1)$. The incoming request rate at time $t - 1$ is $\lambda_m(t-1)$. At time t, the new incoming load is described by the vector of request rates $\lambda_m(t)$. In this type of problem, the decision variables concern the changes in the existing deployment to satisfy the new requirements. To this aim, we describe the new decision variables in Table II.

TABLE II: Additional notation for the dynamic programming model.

Status variables	
$\bar{x}_{m,e}(t-1)$	Allocation of micro-service m to edge e at $t-1$
$\bar{y}_e(t-1)$	Use of edge node e at time $t-1$
Decision variables	
$x^-_{m,e}(t)$	Migration of micro-service m *from* edge e at t
$x^+_{m,e}(t)$	Migration of micro-service m *to* edge e at t
$y^-_e(t)$	Power-*down* edge node e at time t
$y^+_e(t)$	Power-*up* edge node e at time t

The new decision variables $x^-_{m,e}(t)$, $x^+_{m,e}(t)$, $y^-_e(t)$, and $y^+_e(t)$ can be mapped over the new status of the infrastructure as follows:

$$y_e(t) = \bar{y}_e(t-1) - y_e^-(t) + y_e^+(t), \quad \forall e \in \mathcal{E}, \tag{13}$$

$$x_{m,e}(t) = \bar{x}_{m,e}(t-1) - x_{m,e}^-(t) + x_{m,e}^+(t), \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \tag{14}$$

This model also introduces the following additional constraints:

$$\sum_{e \in \mathcal{E}} x_{m,e}^+ = \sum_{e \in \mathcal{E}} x_{m,e}^-, \quad \forall m \in \mathcal{M}, \tag{15}$$

$$x_{m,e}^-(t) \leq \bar{x}_{m,e}(t-1), \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \tag{16}$$

$$x_{m,e}^+(t) \leq 1 - \bar{x}_{m,e}(t-1), \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \tag{17}$$

$$x_{m,e}^+(t) + x_{m,e}^-(t) \leq 1, \quad \forall m \in \mathcal{M}, e \in \mathcal{E}, \tag{18}$$

$$y_e^-(t) \leq \bar{y}_e(t-1), \quad \forall e \in \mathcal{E}, \tag{19}$$

$$y_e^+(t) \leq 1 - \bar{y}_e(t-1), \quad \forall e \in \mathcal{E}, \tag{20}$$

In particular:

- Eq. (13) defines the new power status at the edge nodes as the status of the previous time slot $\bar{y}_e(t-1)$ plus the newly powered-up edge nodes $y_e^+(t)$, minus the powered-down edge nodes $y_e^-(t)$;
- In a similar way the new placement of the micro-services over the edge nodes described in Eq. (14) states that the new allocation $x_{m,e}(t)$ differs from the allocation at the previous time slot $\bar{x}_{m,e}(t-1)$ depending on the micro-services migrations $x_{m,e}^-(t)$, $x_{m,e}^-(t)$;
- Eq. (15) states that a micro-service migrating from an edge node must also migrate to an edge node;
- Constraint (16) states that a micro-service can migrate only to an edge node, where the service is not already allocated, while constraint (17) states that a micro-service can migrate from an edge node only if the service is allocated on that node;
- Constraint (18) states that a micro-service migrating from an edge node must not return to that node;
- Constraints (19) and (20) state that only powered-up edge nodes can be turned off and only powered-down edge nodes can be turned on, respectively.

Other constraints (such as overload and SLA avoidance) are inherited from the problem described in Sec. II-D.

The objective function, instead, re-defines the energy-related objective $P(t)$ as follows:

$$\begin{aligned} P_{dyn}(t) = &\sum_{e \in \mathcal{E}} \big(W_y^+ y_e^+(t) + W_y^- y_e^-(t)\big) + \\ &+ \sum_{\substack{m, \in \mathcal{M} \\ e \in \mathcal{E}}} W_x \big(x_{m,e}^-(t) + x_{m,e}^+(t)\big) + \\ &+ W_p \sum_{e \in \mathcal{E}} P_e(t) \end{aligned} \tag{21}$$

The first part of the objective function concerns the penalty for powering up or down an edge node. The weights W_y^+ and W_y^- are weights introduced to evaluate how much disruption this operation introduces on the infrastructure. We consider two different weight for the two operations as the power on of a node takes typically a longer time and is statistically more likely to fail and cause service disruptions. In a similar way, the second term concerns the disruption due to micro-service migration with its weight W_x. Finally, the third term models the actual power consumption of the infrastructure and weighs equal to W_p.

III. Experimental results

A. Experimental setup

In our experiments, we compare the two previously considered approaches by means of synthetically-generated traces. In our tests, we rely on K-Nitro [6] as the problem solver. The traces are used to create an AMPL model [7] that will represent the problem instance. Each trace comprises 24+1 vectors describing the request arrival rates to the available applications. For the computation of energetic values, we consider each time slot to last for 1 hour so that the whole experiment lasts for 24 hours. For the first time slot $t = 0$ the incoming request rate is randomly distributed across the different applications. Subsequent time slots ($t \in [1, 24]$) are described as variations in the request arrival rate from the previous time slot. The initial problem (that is, for $t = 0$) is always solved using the static algorithm, while for subsequent time slots, the problems are solved using both the static and the dynamic programming approaches described in Sec. II-D and II-E.

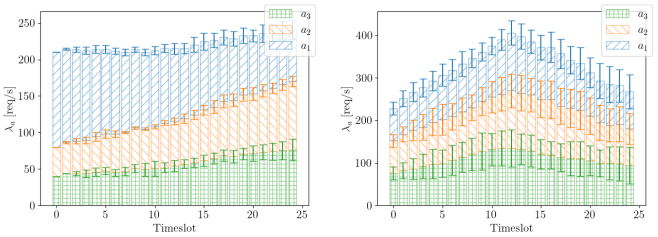
The edge nodes are considered identical in terms of computational power, and we assume their power consumption to range from $P_e^0 = 200$ W when the edge node is idle to $P_e^M = 400$ W for full node utilization. These values are obtained from preliminary tests on a computing node with an Xeon processor, 32 GB of RAM, and 4 disks in RAID configuration.

In order to tune our experimental setup, we considered sets of 3 to 5 applications, where each application is composed of 3 to 6 micro-services. For space reasons, we describe the results only for the case where we have 3 applications ($|\mathcal{A}| = 3$), and each application is composed of 4 to 5 micro-services. For the sake of simplicity, we consider that while micro-services can have a significant variance in the service time, the sum of the service times for each application is constant and equals 10 ms. As a consequence, due to the SLA definition in Sec. II-D, the maximum allowed response time for an application is 100 ms.

We consider two different scenarios for the evolution of request arrival frequencies over the different time slots:

- The *constant-intensity* workload, where the sum of the different workload intensities for the considered applications remains more or less constant over time;
- The *variable-intensity* workload where the global arrival request rate follows a ramp pattern.

The constant-intensity workload is described in Fig. 1a: We observe that some applications (e.g., a_1) have a request rate that decreases through time, while other applications (a_2, a_3)



(a) Constant-intensity (b) Variable-intensity

Fig. 1: Workload intensity

increase the request frequency over the time slots. However, The global workload intensity remains nearly constant throughout the experiment. This means that, while service migrations can occur over the infrastructure, the number of powered-on edge nodes should not change significantly over time.

On the other hand, for the variable-intensity workload (Fig. 1b), the global workload intensity grows steadily up to time slot 12 and then decreases. This means that infrastructure re-configurations that alter the number of active edge nodes are more likely than in the previous scenario.

Over the remaining part of the experimental evaluation, we focus on the output of the optimization problem solver. Specifically, we consider the main performance metrics:

- The energy consumption of edge nodes active at each time slot;
- The average response time of the applications at each time slot;
- The number of re-configurations of the infrastructure (that include micro-service migrations, edge node power-ups, and power-downs).

For the objective function, we consider the weights of the contribution to the overall objective in such a way that $W_p > W_y^+ > W_y^- > W_x$. In particular, an initial round of experiments to tune the problem suggested that the values $\{W_p, W_y^+, W_y^-, W_x\} = \{20, 10, 5, 1\}$ are consistent with the best practices in the management of a distributed infrastructure and ensure a fast convergence of the solvers. Additional experiments with weight within 20% of the considered values provide similar results.

B. Results for constant-intensity workload

As the constant-intensity workload does not require significant changes in the computation power supplied by the infrastructure over time, both the static and dynamic placement can find the optimal number of edge nodes to keep powered on to guarantee that the response time satisfies the SLA (we do not report a figure for this result as it would be trivial). Referring to Tab. III, we observe that the power consumption is close to 26 KWh in the range of 24 hours.

The response time remains below the SLA for every considered timeslot (No figure added for space reasons).

The most notable result is presented in Fig. 2, which shows the number of re-configurations in each time slot. The color

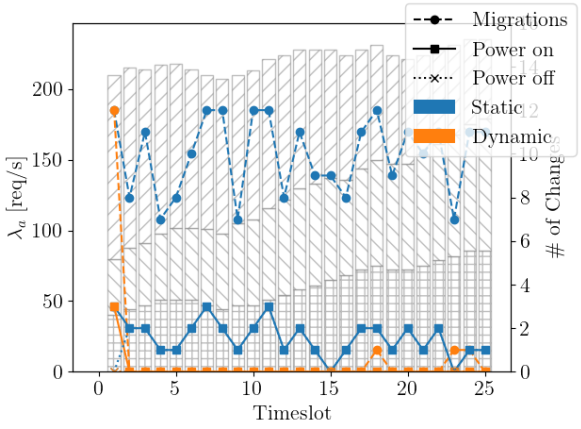


Fig. 2: Re-configurations, constant workload

scheme shows again in blue the static placement approach and in orange the proposed dynamic placement alternative. Different line styles outline the different changes: dashed line with round points for the number of migrations, solid line with square points for power on of new edge nodes; dotted line with crosses for power down of edge nodes.

Observing the number of migrations (dashed line), we notice that the dynamic placement approach can significantly outperform the static alternative with a number of migrations that is reduced by 94% in our experiments: from 249 to 15 over the time slots (a summary is provided in Tab. III, column labeled $\Delta\%$).

In a similar way, it is worth noting that, even if the number of edge nodes powered on is the same for both placement approaches, the static placement turns on and off roughly the same, significant, number of edge nodes at every time slot (the solid and dotted lines are almost overlapped in the graph). Again, referring to Tab. III, we observe that dynamic placement can reduce the number of switches by 94% and can reduce to 0 the number of powered-down edge nodes.

TABLE III: Summary of results

	Static	**Dynamic**	**Δ %**
Energy[Wh]	26076.0	26076.0	0%
Migrations	249	15	94.0%
On	39	3	92.3%
Off	36	0	100.0%

C. Results for variable-intensity workload

We now discuss the variable-intensity workload, where all the applications gradually increase their arrival rate and then start decreasing it at time slot 13. We omit the response time representation for space reason, bu we confirm that also in this case both algorithms can satisfy SLA requirements.

A slightly different behavior between the two approaches is shown when we analyze the energy consumption for the variable-intensity workload (Fig. 1b). In this case, the number of edge nodes changes over time to follow the global request

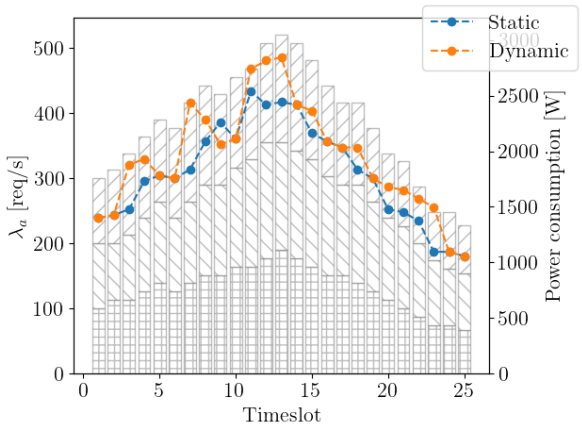


Fig. 3: Power consumption comparison

incoming rate. The static approach is more effective in finding the optimal power configuration on and off-edge nodes that can minimize power consumption. Referring to Tab. IV, the energy consumption for the dynamic alternative is roughly 7.4% higher compared to the static one (48.7 KWh *vs.* 45.1 KWh). However, the power saving comes at the price of a high number of changes in the infrastructure status. Considering the number of power-on, power-off, and migration events, we observe that the dynamic approach can save between 72% and 75% re-configuration operations compared to the static alternative (as pointed out in the column $\Delta\%$ of Tab. IV), confirming that it can significantly reduce service disruption even in a highly dynamic scenario.

TABLE IV: Summary of results

	Static	**Dynamic**	Δ %
Energy [Wh]	45100	48700	7.39%
Migrations	328	81	75.3%
On	59	17	72.0%
Off	56	14	75.0%

IV. Conclusions and Future Work

In this paper, we presented the problem of managing large infrastructures of edge nodes hosting complex applications based on multiple micro-services. In particular, we outline that, as the workload changes over time, the need to re-configure the infrastructure arises. We propose a power-and-performance model for the deployment of multiple co-located micro-services from different applications and we introduce two optimization problems. The first one, namely the static approach, re-computes from scratch at every time slot the optimal placement of the micro-services together with the on-or-off status of the edge nodes. The second approach relies on dynamic programming and takes as input the new workload of the infrastructure and the previous deployment decisions. The model computes the new deployment that satisfies the SLA and powers off unused edge nodes to reduce power consumption but, at the same time, reduces the number of re-configurations to minimize service disruption.

Our tests carried out on different workload scenarios, demonstrate that the dynamic programming approach is quite effective in reducing the amount of re-configurations. The percentage of re-configurations saved is in the order of 72% or more while ensuring a power consumption that is at most 7.4% higher compared to the static alternative.

However, this paper is just a starting point for a new research line. In the future, we aim to carry out additional experiments to evaluate more workload scenarios to better quantify the potential benefits of our proposal. We also aim to study in greater detail the impact of the model parameters, such as the weights of the re-configuration costs. Furthermore, we aim to propose and test heuristics for solving the considered optimization problems.

Acknowledgement

T. Alves de Queiroz thanks the financial support provided by the National Council for Scientific and Technological Development (CNPq) [grant numbers 405369/2021-2, 408722/2023-1, and 315555/2023-8] and the State of Goiás Research Foundation (FAPEG).

R. Lancellotti and M. Iori acknowledge the support of the project ESCALATION (sEcure and SCAlable cLoud bAsed optimizaTION) PR-FESR 2021-2027 Emilia Romagna [PG/2023/303064 - CUP: E97G22000610003].

References

[1] C. Canali, G. Di Modica, R. Lancellotti, S. Rossi, and D. Scotece, "A validated performance model for micro-services placement in fog systems," *SN Computer Science*, vol. 4, no. 4, 2023.

[2] R. Mahmud and A. N. Toosi, "Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework," *IEEE Internet of Things Journal*, vol. 9, no. 6, pp. 4125–4138, 2022.

[3] P. Kayal and J. Liebeherr, "Distributed service placement in fog computing: An iterative combinatorial auction approach," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 2145–2156.

[4] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for qos-driven control of multi-iaas applications," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.

[5] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

[6] *The most advanced solver for nonlinear optimization: KNITRO*, 2024, available at https://www.artelys.com/solvers/knitro/, last accessed on 3rd Aug 2024.

[7] *The fastest optimization toolchains run AMPL*, 2024, available at https://ampl.com/, last accessed on 3rd Aug 2024.