Resource allocation for edge computing with multiple tenant configurations

Andrea Araldo, Alessandro Di Stefano, Antonella Di Stefano

Resource Allocation for Edge Computing with Multiple Tenant Configurations

Andrea Araldo
Télécom SudParis
Institut Polytechnique de Paris
France
andrea.araldo@telecom-sudparis.eu

Alessandro Di Stefano
University of Catania
Department of Electrical, Electronic and Information engineering
Italy
alessandro.distefano@phd.unict.it

Antonella Di Stefano
University of Catania
Department of Electrical, Electronic and Information engineering
Italy
ad@diit.unict.it

ABSTRACT

Edge Computing (EC) consists in deploying computational resources, e.g., memory, CPUs, at the Edge of the network, e.g., base stations, access points, and run there a part of the computation currently running on the Cloud. This approach promises to reduce latency, inter-domain traffic and enhance user experience. Since resources at the Edge are scarce, resource allocation is crucial for EC. While most of the studies assume users interact directly with the Edge submitting a sequence of tasks, we instead consider that users will interact with different Service Providers (SPs), as they currently do in the Web. We therefore consider the case of a Network Operator (NO) that owns the resources at the Edge and must decide how much resource to allocate to the different tenants (SPs).

We propose MORA, a polynomial time strategy which allows the NO to maximize its utility, which can be inter-domain traffic savings, improved users' QoE or other metrics of interest. The core of MORA is that (i) it exploits *service elasticity*, i.e., the fact that services can adapt to the resources allocated by the NO and rely on a remote Cloud for the excess of computation, (ii) it is suitable for *micro-services* architecture, which decomposes a single service in a set of components, which MORA places in the different computational nodes of the Edge and (iii) it copes with *multi-dimensional resources*, e.g., memory and CPUs. After analyzing the properties of the algorithm, we show numerically that it performs close to the optimum. To guarantee reproducibility, the numerical evaluation is performed on publicly available traces from Google and Alibaba clusters and in synthetic scenarios and our code is open source.

CCS CONCEPTS

• **Networks → Cloud computing**; **Network management**; *Programmable networks*; • **Computer systems organization → Cloud computing**; *n-tier architectures*;

KEYWORDS

Cloud Computing, Edge Computing, Resource allocation, Container systems, Network optimization

ACM Reference Format:
Andrea Araldo, Alessandro Di Stefano, and Antonella Di Stefano. 2020. Resource Allocation for Edge Computing with Multiple Tenant Configurations. In *Proceedings of ACM SAC Conference (SAC'20).* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/xx.xxx/xxx_x

SAC'20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Under the paradigm of Edge Computing (EC), computational capabilities, e.g., memory and processing elements, are deployed directly in the access networks, close to the users. This enables low latency applications, reduces the traffic going out from the access networks and can improve user experience. EC is complementary to Cloud [29]. In the Cloud, resources are usually assumed *elastic*, i.e., they are always available, as long the third party Service Provider (SP) is willing to pay. On the contrary, contention emerges in the Edge between SPs sharing limited resources and the problem arises of how to allocate them between SPs. The problem we solve in this work is the one of a Network Operator (NO), owning limited computational resources in its Edge network, which must decide how to distribute them to different SPs. The goal of the NO is to maximize its own utility, which can represent bandwidth or operational cost saving or improved experience for his users [6, 14].

The core of our approach is that we exploit *service elasticity*: a service can run at the Edge under different configurations. Today, these scenarios are common in services as video streaming, in which the SP has to deliver different encodings of the same video and can choose whether to pre-package all these representations and store them, which requires a high amount of memory. Alternatively, with Just In Time Packaging (JITP) SPs can store just few representations and package the missing ones on-the-fly, only when needed, which saves memory space but incurs more CPU usage [14]. In the Edge, we do not have resource elasticity as in the Cloud, but at least we can exploit service elasticity, which is the aim of MORA. We show that, by doing so, the NO can increase its utility with respect to the classical case of one monolithic configuration per SP.

Furthermore, we consider the distributed nature of Edge resources, which can be scattered across different nodes and the fact that services follow a microservice architectural style (Sec. V.B of [25]): a service is composed of different microservices running on *containers*. This allows fine-grained and responsive service adaptivity and resource exploitation, which makes containers attractive for Edge computing [11].

Another important aspect of MORA is its polynomial time efficiency. Since user demand is expected to change in fast time-scales and so are the resources required by the SPs, the allocation must be calculated ideally within few seconds. For example, a Content Delivery Network usually recalculates the association between demand and computing nodes every 10 to 30 seconds [21]. Some work assumes even higher re-allocation frequency [29].

The paper is organized as follows: §2 discuss the related work; §3 describes the architecture we have in mind; §4 reports an Integer Linear Programming (ILP) formulation of the multiple-tenant multiple-configuration allocation problem; §5 describes MORA, its computational complexity and gives a bound to the optimal solution; §6 reports the numerical results on synthetic data and on publicly available traces.

2 RELATED WORK

We study the case of Metro Edge Cloud and Mobile Edge Computing [11, 26], in which there are computation nodes concentrated in small data-centers located into the Network Operator Central Office (CO) or co-located in the base station. While there is vast literature on EC [11, 25, 26], we focus in this section just on work concerning resource allocation. We survey applications of this problem on EC and also on completely different domains, if the applied methodology gives useful insight for our problem. To the best of our knowledge no previous work has attempted to exploit service elasticity (§ 1), which we show instead in this paper to improve resource exploitation in a resource constrained environment like EC. This is, we think, the main merit of our work.

2.1 Resource allocation for container-based EC

The problem of containers resource allocation has been investigated using time-slicing [24], Fuzzy [28], linear programming models [31], reinforcement learning [23]. Others authors also employ non-standard techniques like vertical elasticity [4]. Most work considers only a single type of resource to allocate, i.e., CPU [24], with few exceptions [23, 28]. Some work considers all resources aggregated in one single pool [23], while others [4, 28] consider that they are distributed across different nodes, which complicates the allocation problem. The objective is generally to minimize network traffic, energy [31] or execution time [23].

As for the information to support the allocation decision, most work is based on a "monitor-and-decide" approach [4, 23, 28], but it is becoming common to also use detailed information on the workloads by the users [10, 28]. When implementing allocation strategies, a Docker scheduler [4, 24] is mostly assumed.

2.2 Edge-Cloud hierarchy

EC is complementary to Cloud, i.e., the usual assumption is that a part of service computation is peformed at the Edge and the rest on the Cloud and similarly a part of the required data seats at the Edge and the rest on the Cloud. In a sense, the Edge-Cloud infrastructure is hierarchical [12, 22, 29], where Edge resources are the leaves and the upper nodes are Cloud clusters. In [29] the decision of how much capacity must be provisioned across the levels of the hierarchy. Similar to our proposal, workloads have requirements and can fit into a node if its capacity is not violated, but resources are mono-dimensional. A similar problem is tackled by [22], but nodes are modeled as queues and CPU and network traffic are jointly considered. Queueing models are also employed in [12], which focuses on load balancing between Edge and Cloud. The set up all this work is different from ours, as they do not consider contention between multiple tenants, i.e., the SPs of our set up.

2.3 Other resource allocation problems

Game theory is used to allocate resources between tasks submitted by users [15]. In our work, instead, contention does not emerge between user tasks, but between third party services. Recent literature exists on cache allocation, in which the NO allocates memory to third party SPs to minimize bandwidth consumption [6] or QoS and fairness [9] (CPU is ignored). Similar to our proposal, but in a simpler context, the tasks modeled in [19] can run in different configurations, each using a different combination of resources and resulting in a different perceived utility for the users. Their model explicitly represent the relation between resource usage, an indication of the "quality" achieved and some "utility". However, in their numerical experiments simulation input datas are randomly generated. We thus preferred to associate a certain resource usage to a utility for the Network Operator directly, assuming this relation can be obtained by measurements (of bandwidth consumed, of users' QoE [8]). Moreover, [19] do not consider multiple-servers and multiple-containers. Authors of [30] assume users send a sequence of tasks and each can run under different configurations, requiring a combination of different resource types. They assume users want to run as many tasks as possible and their utility is number of tasks run. While this task-centric vision is more suitable for Grid-like environments, we instead adopt the assumption services and users behave like in current applications in the web environment: instead of submitting task, users instantiate connections with services and use them along a span of time. Therefore, while in [30] resources are consumed every time a task is submitted, we instead assume, as in current Internet services, that resource consumption is not tight to the single task submitted. Take, for example, the case of a video streaming service which requires memory to cache the most popular videos: memory is consumed in a "persistent" way, i.e., independent of the single task submitted by users. We assume however that SP declares the resources needed based on its predicted demand. To summarize, differently from [30], our resource consumption does not come from single user's tasks, but from the needs of SPs. Furthermore, utility is not the one of users, but the one of the NO. Therefore, while [30] maximize the user utility or the product of users' utility, we maximize instead the NO utility, as the NO invested for Edge resources and wants to capitalize them. As a consequence, our problem is different and requires a different strategy.

3 ARCHITECTURE AND INTERACTIONS

3.1 Limits of current architectures

Edge Computing is already being employed by big players in the Internet. As an example, Netflix deploys its own hardware, called Open Connect Appliances (OCAs) [3], into Internet access networks and serves a fraction of users' traffic directly from there. This generates a utility to the NO, in terms of inter-domain traffic saving. On

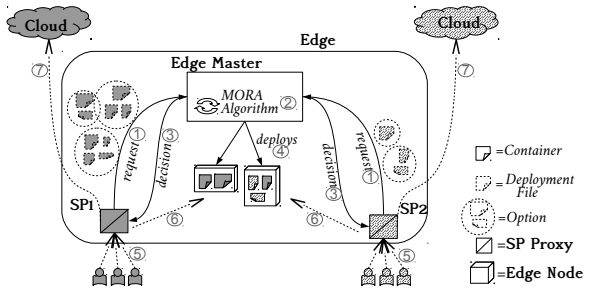

Figure 1: Architecture of MORA .

the other hand, Netflix has all its business assets (content and user information) in its boxes and does not need to share it with NOs. The limit of this solution is its limited *permeability*: it is unfeasible in terms of cost and physical space to install hardware appliances to the very edge of the network, i.e., in many base stations, central offices, access points, etc. Moreover, many other SPs, in addition to Netflix, would benefit from having their hardware installed in access networks, but it would be impractical for NOs to install them all. Furthermore, SPs with less bargaining power than Netflix would have no strength to convince NOs to deploy their hardware appliances. These limits can be overcome if appliances are virtualized, as it is already done in Cloud environments. We thus propose that an NO deploys computational resources at the edge, e.g., memory and processing units, and then allocates slices of them to several third party SPs. The SP can then use its assigned slice as it were a dedicated hardware. Memory encryption technologies [7] can guarantee that data and processing remain inaccessible to the NO, even if they run in its premises. The problem we investigate in this paper is how to allocate resources to each SP in this setting. Note that, while big players may continue to use their hardware appliances, our virtualized solution is probably the only way small or medium SPs can reach the Edge of the network.

3.2 MORA architecture

The goal of MORA algorithm is to choose, for each SP, one of the possible configuration options in which the service can run at the edge. We show here how our framework could be deployed in Edge Computing. The entities and the interactions taking place periodically, i.e., every 5 minutes, between them are depicted in Fig.1. Edge resources are handled by an *Edge Master*, e.g., a Kubernetes Master, managed by the NO. One *SP Proxy* runs for each SP, which sends ① a request to the Edge Master, similar to Kubernetes Deployment files. A request reports different configuration options at which the SP can run its service at the Edge. In particular, an option is a set of containers. Therefore, in the request the requirements, e.g. memory and CPU, of all containers are reported. The request also reports the utility associated to each option, e.g., the bandwidth saved by that configuration option with respect to the case where the SP has no containers running at the Edge. The Edge Master collects the requests from the different SPs; runs MORA algorithm ② to select an option for each SP; communicates ③ the decision to all SP Proxies; and deploys ④ the containers of the chosen options, using the appropriate container images, pre-stored in a repository.

Table 1: Summary of the notation.

Parameters	
M	Number of nodes
N	Number of service providers
J^i	Number of options by SP i
$Z^{i,j}$	Number of containers for option j of SP i
$c_{l,m}$	Amount of resource l in node m
$w_{l,z}^{i,j}$	Amount of resource l required by the container z of option j by SP i
$u^{i,j}$	Utility given by choosing option j of SP i

Decision variables	
$x^{i,j}$	Binary variable, 1 if the option j by SP i is chosen, 0 otherwise
$y_{z,m}^{i,j}$	Binary variable, 1 if the container z of option j by SP i runs on node m, 0 otherwise

Connections from users to a certain SP are intercepted ⑤ by the SP Proxy [20], which decides whether to associate users locally to the Edge ⑥, if the resources of the deployed containers are sufficient. Otherwise, the users are associated to a remote Cloud ⑦, similarly to [12].

4 SYSTEM MODEL

We consider the case of a Network Operator (NO) owning an Edge Computing infrastructure, composed of $m = 1, \ldots, M$ nodes. Resources are of type $l = 1, \ldots, L$. In the numerical results we will consider $L = 2$ resource types, namely memory and processing, which are considered by modern orchestration frameworks as Kubernetes.[1] Each node m has a capacity $c_{l,m}$, which is the amount of resource of type l available. We have $i = 1, \ldots, N$ services competing to use the resources available at the edge. Similar to [13], we consider that there is no unique way to run a service at the edge. If abundant resources are available, a service can be configured in order to exploit them all, thus almost completely running at the edge. If less resources are available, the service may configure itself so to adapt to those and to move some of the computation and data to some remote servers or cloud computing infrastructures (§ 1). We represent this possibility by specifying different configurations (or *options*) $j = 0, 1, \ldots, J^i$ for the same service. In short we will denote with ij the j-th option of service provider i. We assume services are "containerized" [25]. Therefore, each configuration j is composed of a set of containers $z = 1, \ldots, Z^{i,j}$, each of which requires $w_{l,z}^{i,j}$ units of resource type l. The multiple configurations in which a Service Provider (SP) can run its service at the edge denotes its capability to adapt to different amounts of resources available. Each configuration results in a certain utility $u^{i,z}$ for the Network Operator (NO), which in the simplest case represent bandwidth saving [6], which is what we consider in our results with Alibaba cluster traces. Utility can in genral be cost savings [14], QoS or fairness [9], elaboration time savings [15], depending on the application and the information available. As commonly done in

[1] https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/#resource-types

the literature [9, 14, 16], we adopt a "snapshot" approach by assuming that the resources needed for the configurations and the other characteristics of the configurations are known at the moment of taking the resource allocation decision. In our case, we assume they are declared by the SP itself, which is the only one knowing exactly the algorithms and the data involved in its computation. This is in line with today containerized environments. For example, in Kubernates it is possible to define memory, CPU and bandwidth limits when Deployment files are submitted. As commonly done in the literature [16, 21] we assume that there are mechanisms able to provide good estimates of resources and utility, which fall outside the scope of this paper. Note also that we do not consider the cost of instantiating and realising containers, since the snapshot approach cannot capture this kind of dynamics. We plan to fill this gap in future work.

The decision of the NO about which option from each SP should be accepted in the Edge and where to place the correspondent containers can be formulated in the following Integer Linear Program (ILP) as proposed in a working paper [5]. The binary decisions variable are $x^{i,j}$, which is 1 if the j-th option of the SP i is chosen, and $y^{i,j}_{z,m}$, which is 1 if the z-th container of the j-th option of SP i is placed on node m.

The symbols used in the paper are reported in table 1.

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N_i} u^{i,j} \cdot x^{i,j} \tag{1}$$

$$s.t. \sum_{m=1}^{M} y^{i,j}_{z,m} = x^{i,j} \qquad \begin{matrix} i = 1 \dots N \\ j = 1 \dots J^i \\ z = 1 \dots Z^{i,j} \end{matrix} \tag{2}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{J^i} \sum_{z=1}^{Z^{i,j}} y^{i,j}_{z,m} \cdot w^{i,j}_{l,z} \le c_{l,m} \qquad \begin{matrix} l = 1 \dots L \\ m = 1 \dots M \end{matrix} \tag{3}$$

$$\sum_{j=1}^{J^i} x^{i,j} \le 1 \qquad i = 1 \dots N \tag{4}$$

$$x^{i,j}, y^{i,j}_{z,m} \in \{0, 1\} \qquad \begin{matrix} i = 1 \dots N \\ j = 1 \dots J^i \\ z = 1 \dots Z^{i,j} \\ m = 1 \dots M \end{matrix} \tag{5}$$

The objective is to maximize the utility (1), setting the binary variables $x^{i,j}$. Constraints (2) guarantee that each container z of the chosen option j of SP i ($x^{i,j} = 1$) is deployed ($\exists m \in \{1 \dots M\}$: $y^{i,j}_{z,m} = 1$). Constraints (3) guarantee that the sum of the requirements for the set of containers deployed on the same node m for each resource l is less than the total amount of available resources in node m so that these containers can actually run on the node. Finally, constraints (4) ensure that a SP can deploy at most one option in the Edge cluster.

Proposition 1. *Problem P is NP-hard.*

Proof. P reduces to a Knapsack Problem with $L = 1, M = 1, J^i = 1, i = 1, \dots, N$ and $Z^{i,j} = 1, i = 1, \dots, N; j = 1, \dots, J^i$, which is NP-hard. □

Reducing to 1 some of the dimensions L, M, J^i, Z, we can reduce the problem to Set-union Knapsack, Multiple-Choice Knapsack or Knapsack Problems. Our problem is complex and we rule out the possibility to construct Fully Polynomial Time Approximation Schemes, as §9.4.1 of [17] shows that they cannot exist (unless P=NP), already for the simpler case of $M = 1, Z^{i,j} = 1$ and $J^i = 1, i = 1, \dots, N$, which is known as l-KP. All we can do is then to propose a heuristic and show it is close to the optimum numerically.

5 MORA

We now introduce MORA, our proposed resource allocation strategy.

5.1 Preliminary definitions

The MORA heuristic uses aggregate values for the resource requirements and availability, in order to neglect, at a first stage, the complexity represented by the fact that resources available are scattered across different nodes, resource required are split in different container requirements and requirements are multi-dimensional.

To this aim, we need to define the overall resource requirements of an option j of a SP i as

$$w^{i,j}_l = \sum_{z=1}^{Z^{i,j}} w^{i,j}_{l,z}. \tag{6}$$

We introduce a number $h_l \ge 0$, that we call "relevance value", since its role is similar to the relevance values in §9.5.1 of [17]. We also define the generalized resource utilization of an option j of a SP i as:

$$w^{i,j} = \sum_{l=1}^{L} h_l \cdot w^{i,j}_l \tag{7}$$

To ease computation, MORA heuristic algorithm does not consider all the possible options, but it first removes the *dominated options* and then *LP-dominated* options, defined as follows, which do not provide significant utility gain with respect to the resources they require.

Definition 1. *For any SP i, an option j is dominated by another option $j' \neq j$ iff (i) $u^{i,j'} > u^{i,j}$ and $w^{i,j'} \le w^{i,j}$ or (ii) $u^{i,j'} \ge u^{i,j}$ and $w^{i,j'} < w^{i,j}$. An option is dominated, if it is dominated by some other option.*

Before giving the definition of LP-dominance, we need to define the *efficiency* of a *jump* as follows.

Definition 2. *For a service provider $i = 1, \dots, N$, the efficiency of a jump $j \to j'$, where $w^{i,j'} > w^{i,j}$ is:*

$$e^{i,j \to j'} = \frac{u^{i,j'} - u^{i,j}}{w^{i,j'} - w^{i,j}} \tag{8}$$

Definition 3. *A non dominated option j of a SP i is LP-dominated, if there exist other non dominated options j', j'' such that $u^{i,j'} < u^{i,j} < u^{i,j''}$, $w^{i,j'} < w^{i,j} < w^{i,j''}$ and $e^{i,j' \to j''} \ge e^{i,j' \to j}$ The option j is an LP-extreme if it is neither dominated nor LP-dominated.*

The names "LP-dominance" and "LP-extremes" come from the fact that the concept is related to the LP-relaxation of the Mutliple Choice Knaspack Problem, but this is not relevant for our scope. Fig. 2 illustrates the concept of LP-extremes, similarly to Fig. 11.1

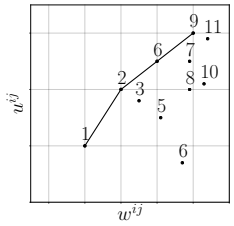

Figure 2: Example of set of options of a SP i. The options connected by the line constitute the ordered list of LP-extremes.

of [17]. We order the LP-extreme options in the list defined as follows.

Definition 4. *For each SP i, we denote with $\underline{j}^i$ the list of its LP-extreme options. We denote with $\underline{j}^i[k]$ the option at its k-th position. This list is ordered in increasing values of $w^{i,j}$, such that $w^{i,j^i[k]} \leq w^{i,j^i[k+1]}$. In the first position of such list we add a fictitious "null" option, such that $w^{i,j^i[0]} = u^{i,j^i[0]} \triangleq 0$.*

Proposition 2. *For any SP i, the list $\underline{j}^i$ of LP-extremes can be computed in $O(|J^i| \cdot R^i)$, where R^i is the number of LP-extremes.*

Proof. Ch. 11 of [17] shows that LP-extremes correspond to the convex hull of the set of options $j = 1, \ldots, J^i$. To compute the convex hull we use [18], which has the complexity above. □

5.2 MORA Algorithm

The Multiple Option Resource Allocation (MORA) algorithm is shown in Algorithm 1. It takes as input the set of parameters of the ILP (9)-(4) describing the scenario plus a configuration parameter h_l for $l = 1, \ldots, L$. The algorithm returns a solution, i.e. values for any decision variable. The algorithm solves two decision problems that the NO must solve: (i) *Option selection*: which option (or configuration) per SP must be accepted (variables $y^{i,j}_{z,m}$) and (ii) *Container placement*: in which Edge nodes we should place the containers of the selected options (variables $y^{i,j}_{z,m}$). The pseudo code of Alg. 1 is mainly devoted to option selection and calls Alg. 2 for the container placement.

5.2.1 ***Option selection.*** MORA is iterative. In each iteration, each SP i has a *current position* k^i, which corresponds to the option $j^i[k^i]$. Each SP i has also a *jump efficiency* E^i (line 6), which denotes the efficiency achieved when advancing its position, i.e., the utility gain obtained going from option $j^i[k^i]$ to $j^i[k^i+1]$ divided by the additional generalized resource utilized. Observe that $u^{i,j^i[k]} < u^{i,j^i[k+1]}$ and $w^{i,j^i[k]} < w^{i,j^i[k+1]}$ by construction, and thus $E^i > 0$.

Then, in each iteration t, we select a SP and we check whether we can change its current option $j^i[k^i]$ to $j^i[k^i+1]$. We say that service provider i performs a *jump* $j^i[k^i] \rightarrow j^i[k^i+1]$. As one can expect, we select the SP whose jump efficiency is the highest (line 12). We call this SP the *jumping* SP of iteration t, as it is the one that changes option (the options of the other SPs remain unchanged).

We then try to place the containers of the jumping SP i^* (line 13). If we succeed, we advance its current option, thus allowing i^* to jump from $\underline{j}^{i^*}[k^{i^*}]$ to $\underline{j}^{i^*}[k^{i^*}+1]$ (line 15). Otherwise, we remove the option $j^{i^*}[k^{i^*}+1]$ that we have not been able to place. We update the jump efficiency of i^*.

The algorithm terminates when the lists $\underline{j}^i$ of all the SPs have been visited (line 10).

5.2.2 ***Container placement.*** The placement operations are described in Alg. 2. We will refer to *placement* as a mapping of a container to an edge node. The algorithm works by constructing a *tentative placement* $\hat{y}^{i,j}_{z,m}, \forall i, j, z, m$. If we are able to construct a feasible tentative placement, i.e., we are able to place all the above-mentioned containers in the available nodes without violating the resource constraints, we update the actual placement $y^{i,j}_{z,m}$ accordingly (Line 21). Otherwise, we ignore the tentative placement and we leave the actual placement unchanged.

The tentative placement is practically identical to the actual placement (Line 1), except for the containers of the jumping SP i^*. Since we want to place the containers of the new option j^* of SP i^*, we first reset all its previously selected options (Line 2). Then, we iterate through the containers of option j^* of SP i^*, and we try to place them one by one. In order to place a container z, we first check what are the nodes $\mathcal{M}(z)$ whose residual capacity is enough to host it (Line 7) and we chose one of them (Line 10). Similarly to Sec.III.C of [27], this choice is based on the product of residual capacities, but we use arg max while [27] chooses arg min. Since the performance of our algorithms are already good with this current rule, we defer the exploration of other rules in future work.

To summarize, at each iteration we take a hierarchical decision: we first select an option of a service provider, based on the best jump concept. Then, we try to place the composing containers in the available nodes. Note that the operations within each iteration does not correspond to any change to the actual resource allocation. The algorithm is always executed until the terminating condition, and only after that the result is taken to decide the actual resource allocation.

5.3 Properties of MORA

We now characterize MORA in terms of time complexity, we find an upper bound of the problem and we discuss the impact of the algorithm parameters h_l.

5.3.1 ***Computational Complexity.*** MORA is a polynomial time algorithm. We omit the following proof for lack of space.

Proposition 3. *The time complexity of MORA is $O(N^2 JRZML))$, where J is the maximum amount of options per SP and Z is the maximum number of containers per option and R the maximum number of LP-extremes per SP.*

5.3.2 ***Upper bound.*** Knowing the upper bound of P is important, since we can compare it with the utility provided by the MORA heuristic and verify how far it is from the optimum. Moreover,

Algorithm 1 MORA algorithm.

Input: $u^{i,j}$, $w^{i,j}_{l,z}$, $c_{m,l}$, h_l.
Output: $x^{i,j}$, $y^{i,j}_{z,m}$, upper bound $\hat{u}$.
// *Initialization*
Set $x^{i,j} := y^{i,j}_{l,m} := 0$ for all l, m and all options i, j
for all SP $i := 1, \ldots, N$ **do**
  Compute $w^{i,j}$, $j = 1, \ldots, J^i$, as in (7).
  Compute the ordered list $\underline{j}^i$ of options of SP i as in Def. 4.
  Initialize the current position $k^i := 0$ on such list.
  Compute

$$E^i := \begin{cases} e^{i,\underline{j}^i[k^i] \to \underline{j}^i[k^i+1]} & \text{if } k^i + 1 \neq \text{ end of the list} \\ -\infty & \text{otherwise} \end{cases}$$

end for
// *Main loop*
for Iteration $t := 0, 1, \ldots$ **do**
  if $E^i = -\infty$ for $i := 1, \ldots, N$ **then**
    break // We arrived at the end of all lists $\underline{j}^i$.
  else
    $i^* := \arg\max_i E^i$ // *Jumping SP*
    success := placeContainers(i^*, $\underline{j}^i[k^i] + 1$) // *see Alg. 2*
    if success = True **then**
      $k^{i^*} := k^{i^*} + 1$ // *Advance current option*
    else
      Remove the $k^{i^*} + 1$-th element of the list $\underline{j}^{i^*}$.
      // Note that, now the option that was in the
      // $k^{i^*} + 2$-th position (if any), now goes to the
      // $k^{i^*} + 1$-th position.
    end if
    Update

$$E^{i^*} := \begin{cases} e^{i^*,\underline{j}^{i^*}[k^{i^*}] \to \underline{j}^{i^*}[k^{i^*}+1]} & \text{if } k^{i^*} + 1 \neq \text{ end of the list} \\ -\infty & \text{otherwise} \end{cases}$$

  end if
end for
//Translate to ILP notation
Set $x^{i,j} := 1$ for $j = \underline{j}^i[k^i]$ if $k^i > 0$, for any SP i.
return $x^{i,j}$, $y^{i,j}_{z,m}$.

Algorithm 2 Container placement algorithm.

Input: i^*, j^*
Output: boolean *success*.
$\hat{y}^{i,j}_{z,m} := y^{i,j}_{l,m}$, $\forall z, j, l, m$
// Release the containers of the current option of i^*:
$\hat{y}^{i^*,j}_{z,m} := 0$, $\forall j, z, m$
// Compute the residual capacity given by the tentative placement:
$\hat{c}_{l,m} := c_{l,m} - \sum_{i=1}^{N} \sum_{j=1}^{J^i} \sum_{z=1}^{Z^{i,j}} \hat{y}^{i,j}_{l,m} \cdot w_{l,z}$
success := True
for all $z := 1, \ldots Z^{i^*,j^*}$ **do**
  // See which nodes can host container z:
  $\mathcal{M}(z) := \{m \in \{1, \ldots, M\} | w^{i^*,j^*}_{l,z} < \hat{c}_{m,l}, l = 1, \ldots, L\}$
  if $\mathcal{M}(z) \neq \emptyset$ **then**
    // Select one of those nodes:
    $m(z) := \arg\max_{m \in \mathcal{M}(z)} \prod_{l=1}^{L} \hat{c}_{l,m}$
    $\hat{y}^{i^*,j^*}_{z,m(z)} := 1$ // Assign the container to the selected node
    $\hat{c}_{l,m} := \hat{c}_{l,m} - w^{i^*,j^*}_{l,z}$ // Update the residual capacity
  else
    // It is not possible to place container z,
    // and thus the entire option
    success := False
    break
  end if
end for
if success = True **then**
  // The tentative placement is accepted as actual placement
  $y^{i^*,j^*}_{z,m} := \hat{y}^{i^*,j^*}_{z,m}$, $\forall z, m$
end if
// Else, we leave the actual placement unchanged
return success

MORA is anytime, i.e., if we terminate it at any iteration, it returns a valid allocation. The distance from the upper bound can guide us in the decision whether to continue the iterations or not, which can potentially save computation time. In order to do so, we fisrt fix any values for $h_l, l = 1, \ldots, L$, compute $w^{i,j}$ as in (7) and $c_{\text{tot}} \triangleq \sum_{m=1}^{M} \sum_{l=1}^{L} h_l \cdot c_{l,m}$. Then, we resort to a problem known in the literature as Multiple Choice Knapsack Problem (MCKP):

$$\max \sum_{i=1}^{N} \sum_{j=1}^{N_i} u^{i,j} \cdot x^{i,j} \qquad \text{(MCKP)} \quad (9)$$

subject to

$$\sum_{i=1}^{N} \sum_{j=1}^{J^i} x^{i,j} \cdot w^{i,j} \leq c_{\text{tot}}; \quad \sum_{j=1}^{J^i} x^{i,j} \leq 1; \quad x^{i,j} \in \{0,1\} \quad \begin{matrix} \text{l} = 1 \ldots \text{L} \\ \text{i} = 1 \ldots \text{N} \\ \text{j} = 1 \ldots \text{J}^i \end{matrix} \quad (10)$$

Since a solution that satisfies (2)-(4) also satisfies (10), the optimal solution of MCKP is an upper bound to the optimal solution of P. We resort to an algorithm from Dyer and Zemel (Fig. 11.5 of [17]) that computes in linear time the optimal solution of the LP-relaxation of MCKP, which is an upper bound of MCKP, and thus is an upper bound of our original problem P. We can thus claim the following

Proposition 4. *An upper bound $\hat{u}$ of the original problem P can be found in $O(\sum_{i=1}^{N} J^i)$.*

5.3.3 ***Impact of the relevance values.*** The relevance values $h_l, l = 1, \ldots, L$ are algorithm parameters that change the results we obtain. Indeed, if we change h_l, the values of $w^{i,j}$ change for all js (see (7)) and thus the list $\underline{j}^i$ changes as well. This value serves to weight resource types among them. If, for example, a certain resource type l, say memory, is scarce in the Edge, we should tend not to select options that consume a lot of resource l. This can be achieved by setting a high value of h_l. By doing this, an option i, j that consumes a lot of l-resource would have a high $w^{i,j}$, and thus would have less chances to be in the LP-extremes list $\underline{j}^i$ (it would tend to be on the right of Fig. 2). Moreover, jumping from another option j' to j would likely result in a low efficiency $e^{i,j' \to j}$ and Alg. 1 would prefer other jumps. Observe also that different values of h_l would result in different upper bounds $\hat{u}$. In this way, one can compute different upper bounds and just consider the minimum value.

6 NUMERICAL RESULTS

We evaluate MORA in (i) synthetic scenarios that we construct ourselves and on (ii) publicly available traces from Google and Alibaba clusters. While (i) allow to study the sensitivity of MORA to selected parameters, (ii) allow to assess performance in real-world cases. We compare MORA to the *optimal solution* (computed via (1)-(4) using GLPK) and to a *Naive strategy*. The latter iterates over the available SPs and for each one chooses a random option to be deployed. It then tries to deploy each container of the chosen option in the first node that fits the requirements of the container itself. Whenever the first SP cannot be placed in the Edge cluster the naive algorithm stops. The bad performance that will be shown for Naive demonstrates that is important to select the "right" option per SP and the "right" node per container. In all plots, we keep all the parameters at their default values (Tab. 2) and we make vary only the parameter(s) explicitly specified.

In what follows we study the computation time (§6.1.1), the utility achieved and the resources left unused after the allocation (§6.1.2-6.1.5). We also study how resources are distributed among SPs (§6.1.6). In the real traces results, we show the achieved utility varies with number of SPs (§6.2.1). Since MORA is an anytime algorithm, we report how the utility evolves during its iterations (§6.2.2).

All results on synthetic scenarios are averaged across 20 runs and 95% confidence intervals are reported, which may not be visible when they are too small. They are calculated on a Intel Xeon CPU E5-4610 v2 @ 2.30GHz with 256GB RAM. The model of the ILP in glpk and the python code of MORA are available as open-source on GitHub [2].

6.1 Results on synthetic scenarios

We consider an Edge Cloud [26] consisting of M identical Intel Xeon nodes with 4 sockets and 4 cores with hyper-threading enabled. Therefore we can consider each node with 16 cores hyper-threaded and we associate $32GB$ RAM to each of them. For each scenario we consider N SPs, each declaring the same number J of configuration options.

Each configuration option is described in terms of the required Z containers. The memory and the processing required by a container z of the j-th option of service i are drawn from uniform random distributions with mean $\bar{w}_l$, with $l = \{RAM, CPU\}$. They are expressed as dimensionless values for CPUs while the memory is expressed in GB. A fractional value of CPU is to be interpreted as fraction of CPU time. For each scenario, the two values $\bar{w}_{\text{RAM}}$ and $\bar{w}_{\text{CPU}}$ are calculated as follows. First, a *load factor* K is chosen, and then $\bar{w}_l$ computed as

$$\bar{w}_l \cdot Z \cdot N = K \cdot c_{l,\text{tot}}; l = \{\text{CPU,RAM}\} \tag{11}$$

where $c_{l,tot} = \sum_{m=1}^{M} c_{l,m}$ is the total amount of resource of type l available at the edge. In other words, on average we allow services to request K times the available resources.

The default values are reported in Table 2. In all the following plots, we will make only a subset of parameters vary and keep the others at their default value. As in [13, 19], the utility associated to each option is a random variable in these synthetic scenarios

[2] https://github.com/aleskandro/cloud-edge-offloading

Table 2: Default values of the reference scenario evaluated

Number of SPs	N	50
Number of nodes	M	8
Number of options	J	5
Number of containers	Z	8
Load factor	K	1.8

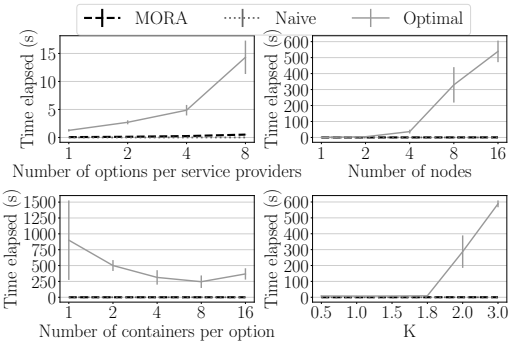

Figure 3: Time to compute solutions for ILP, MORA and Naive strategies

(it will instead be a real value directly taken from the traces in the Alibaba case). Moreover, we follow the common assumption in the literature [6, 9] that there is a concave relation between the resources used and the utility, which results in a diminishing return: the more resources are used by a SP configuration, the larger one should expect the utility to be, but the additional utility tends to decrease with the resources. Using the notation (6) for $w^{i,j}_{\text{CPU}}$, $w^{i,j}_{\text{RAM}}$, the utility is the following function of the required resources:

$$u^{i,j} = \alpha^{i,j} \cdot \left(\frac{w^{i,j}_{\text{CPU}}}{c_{\text{CPU,tot}}} \right)^{\frac{1}{\beta^{i,j}_{\text{CPU}}}} + (1 - \alpha^{i,j}) \cdot \left(\frac{w^{i,j}_{\text{RAM}}}{c_{\text{RAM,tot}}} \right)^{\frac{1}{\beta^{i,j}_{\text{RAM}}}} \tag{12}$$

where $\alpha^{i,j}, \beta^{i,j}_{\text{CPU}}, \beta^{i,j}_{\text{RAM}}$ are sampled, for each option, from random uniform distributions between 0 and 1 for $\alpha^{i,j}$ and between 1 and 5 for $\beta^{i,j}_{\text{CPU}}$ and $\beta^{i,j}_{\text{RAM}}$. Since these parameters are random variable, (12) "loosely" show monotonicity and concavity, but is not exactly a monotone and concave function. We did this on purpose since: (i) in realistic scenarios this relation may not be as "clean" as assuming a perfectly increasing and concave function; (ii) we want to check the performance of our solution in pessimistic and 'unclean" situations. Note that our construction follows the assumptions usually adopted in the literature [6, 9, 13, 19]. We will not need these assumptions anymore when working with Alibaba traces.

Note that, for all feasible options, $u^{i,z} \in [0, 1]$. Since a feasible solution selects at most one option per SP, we can be sure that $u^{\max} := N$ is an upper bound to u^{tot}. We define the *overall normalized utility* as $u = u^{\text{tot}}/u^{max}$ eqnnum .

By slight abuse of terminology, in what follows we will shortly refer to "utility" to indicate the overall normalized utility.

6.1.1 Time efficiency. Fig. 3 shows that the computation of the Optimum from the ILP is too slow for the allocation frequency envisaged in practical deployments, as discussed in §1. On the

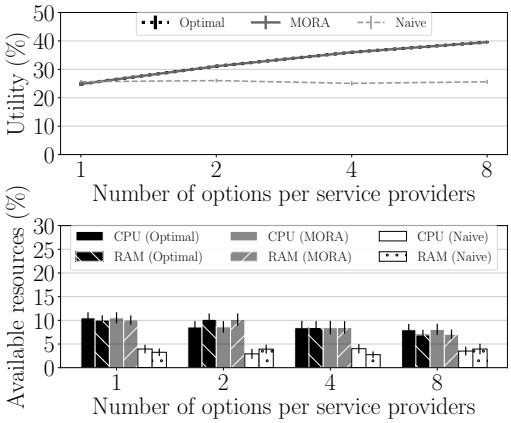

Figure 4: Benefits of multiple options.

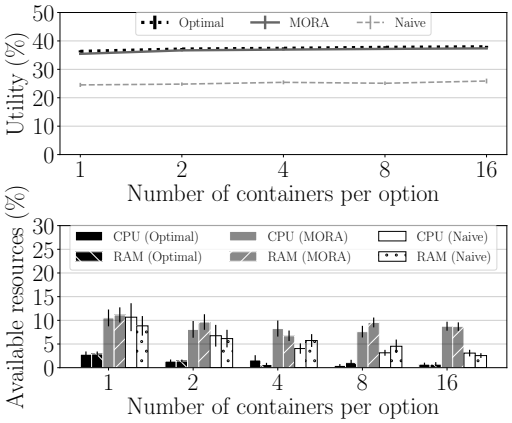

Figure 5: Effect of containerization.

contrary, MORA remains within 0.05s, as the Naive policy, while also achieving almost optimal utility, as next sections will show..

6.1.2 Benefits of multiple options. Fig. 4 shows that utility increases with the number of options per SP. Note that the classic assumption corresponds to the first point of the plot, SP=1. While varying the number of options from 1 to 8 the utility has a gain almost equal to 60%, which would be lost with classic approaches and which instead we can grasp by exploiting service elasticity. This is equivalent to virtually increase the available resources, by just using them better. Observe also that MORA uses resources as the optimum, while Naive, despite providing poor utility, uses ~3.3 times more resources than optimal/MORA).

6.1.3 Insensitivity to scaling. We verified that, if we increase the number of nodes available (thus increasing the overall resources) and we increase accordingly the the option requirements, in order to always keep a load factor K equal to the default value, the utility is not affected. We omit the plot for lack of space. This tells us that the results presented here are likely to be consistent even when scaling the problem and would hold on tiny instances of EC as well as in larger EC clusters.

6.1.4 The more you containerize, the better the utility. Fig. 5, reports for MORA a 11% increase of utility when providing a service through a set of micro-services [25] running on different containers instead of a single one. Indeed, keeping the same overall resource requirements, "smaller" containers are easier to place into Edge nodes.

6.1.5 Effects of load (Fig. 6). Increasing the load K (11), we are increasing the amount of resource contention among SPs. Recall that $K = 0.5$ denotes requests with overall resource requirements that are half of the available resources. In this case, the highest utility-option of each SP is likely to be satisfied. For $K \leq 2$, the more the K, the more utility, from 28% to 40%. This is expected since (i) increasing the K, we are increasing the resource requirements of each option and (ii) the utility of each option is randomly generated as an increasing function (12) of the resources. However, if the

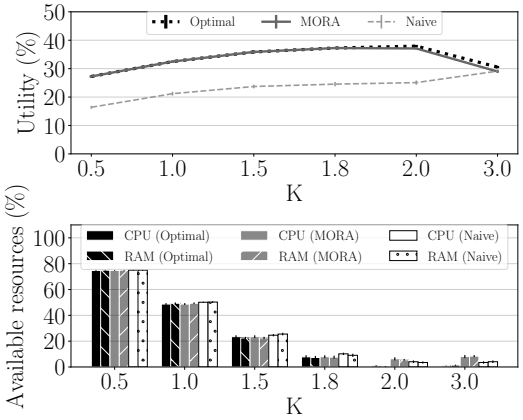

Figure 6: Effects of load.

load factor increases more than 2 the utility starts to decrease both for the ILP and MORA and equals the naive policy. Further investigation is required to explain this behavior in our future work. Hypothesis are: (i) there is a concave relation between resources and utilities (see (12)), which reflects in the diminishing returns observed when increasing K, and the the resources utilized; (ii) as resources demanded by the containers become larger as K increases, it is more difficult to place them, which is confirmed by the fact that the overall resources used with $K = 2$ and $K = 3$ remain unchanged. From these results, at least in our scenarios, we observe that after a certain load threshold the network operator should increase the Edge resources in order to maintain utility levels. As expected, the bottom figure shows that the NO needs to consume more resources to satisfy higher loads.

6.1.6 Distribution of resources and utility among SPs . In this paper we mainly consider that utility benefits NO, but in reality it also benefits SPs. Therefore, it is important to check if resource allocation is fair. Fig. 7 reports the result from one run with 50 SPs.

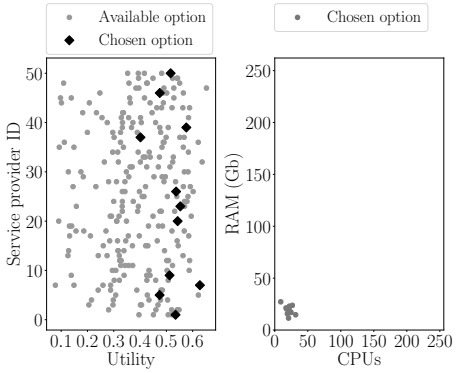

Figure 7: Distribution of resources and utility between 50 SPs. The x and y scale of the right plot are the total available RAM and CPUs.

On the left, the points along each line represent the utility of the different options declared by one SP. The "◆" is the option chosen by MORA. Note that MORA only selects options for 11 SPs, and thus the others do not get any resources since their contribution to the overall utility would not be remarkable. In the right plot, the points are the requirements of the options of the 11 SPs chosen to be deployed in the Edge. From the plot, MORA allocates a similar amount of resources to the selected SPs, which does not necessarily reflect in equality of utilities selected.

6.2 Results on real traces

We now use Google [2] and Alibaba [1] cluster traces. We assume 8 nodes are available, as in table 2.

Google traces includes a list of jobs, each composed by a set of tasks. We consider the requested RAM and CPU associated to each task when the correspondent at time of job's submission. These values are expressed as a fraction of the available resources. To represent a SP i, we randomly select J jobs and we interpret each as an option i, j. Each task composing that job is mapped to a container z. We used (12) to compute the utility of each option.

Alibaba traces include a list of applications, each comprised of several containers sharing the same application index. We map an Alibaba application to an option. As in the Google case, We generate each SP by randomly select a set of options (Alibaba applications). In Alibaba traces, RAM requirements are expressed as percentage of RAM available in one node while CPU requirements are expressed as percentage of usage of one single core. Each machine in Alibaba cluster has 96 cores. We set the nodes in the simulations to reflect these requirements. The trace also reports the bandwidth associated to a container. We calculate the bandwidth of each option as the sum of the bandwidth of the containers of the corresponding application. This is the value of utility that we use. The rational is that we assume that a SP generates a certain traffic toward users. If we select a certain option of a SP, the correspondent bandwidth is served locally at the edge and only the remaining part part must be served by the Cloud, an assumption common in the literature [12].

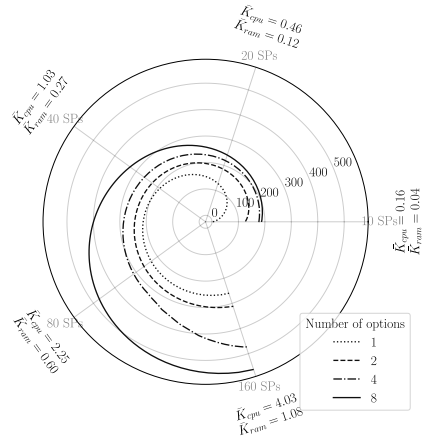

Figure 8: Utility while varying the number of SPs and the number of options provided using Alibaba Traces

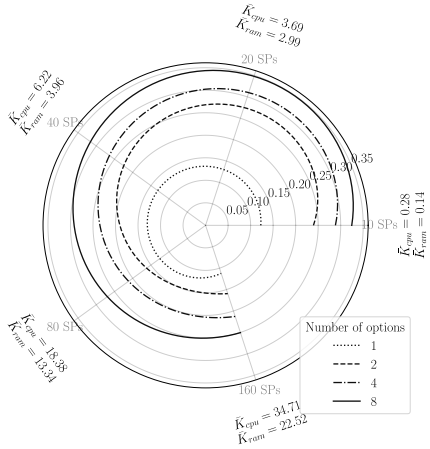

Figure 9: Relative utility while varying the number of SPs and the number of options provided using Google Traces

Therefore, the value of the utility indicates the bandwidth saving in this case.

6.2.1 Effects of number of SPs. Fig. 9 and 8 confirm what was observed in the synthetic scenario. Both in the Google and the Alibaba case the utility increase when exploiting service elasticity increasing the number of options per SP. We also vary the number of SPs considered, which result in an increase in the load, which we quantify with $\bar{K}_{\text{CPU}}$ and $\bar{K}_{\text{RAM}}$, reported in the figure and calculated as:

$$\bar{K}_l = \bar{W}_l \cdot N \cdot \bar{Z} / \sum_{m=1}^{M} c_{l,m}$$

where $\bar{W}_l$ is the average requirement of resource l through all the containers and $\bar{Z}$ is the average number of containers per option, l is $\{RAM, CPU\}$. In the Alibaba figure, as expected, the more SPs join the Edge, the more the utility (bandwidth saving) is achievable. In

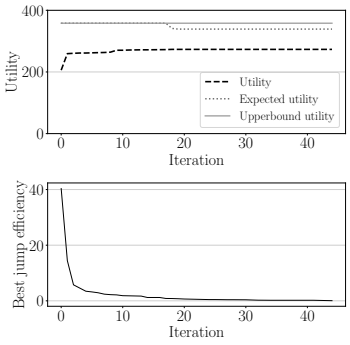

Figure 10: Utility and Best jump efficiency by time

the Google figure, instead, we report the normalized utility (13) (the total utility can be obtained by multiplying it by N). Observe that the values of load K_{CPU} and K_{RAM} are very different. And different are also the Google, Alibaba and synthetic scenarios. However, the benefits of service elasticity consistently show themselves.

6.2.2 MORA as anytime algorithm. We use here Alibaba traces, assuming 8 nodes and 20 SPs. We show in Fig. 10 the utility of the solution computed at every iteration t. By construction, the efficiency E^i of the best jump (bottom plot) is decreasing with t. This ensures that most of the utility is already achieved in the first iterations (top plot) and allows us to prematurely stop the algorithm if the time available to compute the allocation is scarce, still having a good solution at hands. The upper bound to the optimal solution (§5.3.2) reported in the figure also confirms that we are already close to the optimum in the first iterations. By exploiting the fact the monotonicity of E^i we can also easily calculate, at any iteration t, the *Expected utility*, i.e., the utility that we can achieve at most if we continue the algorithm until the end instead of interrupting at t. We omit the details of this calculation for lack of space, but we report it in the figure to show that it can also be a useful guidance to decide whether to stop MORA before its end for a faster result.

7 CONCLUSIONS AND FUTURE WORK

This paper presented MORA, a strategy for resource allocation for Edge Computing (EC), where tenants are third party Service Providers (SPs). The novelty of this work is that it exploits service elasticity: by allowing SPs to declare the different configurations (aka options) in which they can run, we show that the Network Operator (NO) owning EC resources can greatly increase utility. Relying on service elasticity is crucial in resource-constrained environments as EC.

Our future work will be devoted to scenarios where SPs arrive at different times, exploiting a time-batched implementation of the heuristic. We plan to realize a testbed using Docker and Kubernetes. Moreover, the architecture and the heuristic itself have to be expanded in order to take into account different NOs (Edge roaming). We consider using Game Theory to design mechanisms to ensure truthful declaration of options by SPs, which could also be enforced by appropriate resource and utility monitoring. We are also interested in studying allocation strategies in presence of noise in the values of resource usage and utility.

REFERENCES

[1] [n. d.]. Alibaba Cluster Trace Program. https://github.com/alibaba/clusterdata
[2] [n. d.]. Google Borg Cluster usage traces. https://github.com/google/cluster-data
[3] [n. d.]. OpenConnect service by Netflix. https://openconnect.netflix.com/en/
[4] Y. Al-Dhuraibi et al. 2017. Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER. In *IEEE CLOUD*.
[5] Andrea Araldo, Alessandro Di Stefano, and Antonella Di Stefano. 2019. Edge-MORE: Improving Resource Allocation with Multiple Options from Tenants. *arXiv preprint arXiv:1908.01526* (2019).
[6] A. Araldo, G. DÃąn, and D. Rossi. 2018. Caching Encrypted Content Via Stochastic Cache Partitioning. *IEEE/ACM Trans. Netw.* 26, 1 (Feb 2018), 548–561.
[7] Sergei Arnautov et al. 2016. SCONE: Secure Linux Containers with Intel SGX. In *USENIX Symp. on Op. Sys. Des. and Impl.* 689–703.
[8] Francesco Bronzino et al. 2019. Lightweight , General Inference of Streaming Video Quality from Encrypted Traffic. (2019). arXiv:arXiv:1901.05800v2
[9] Weibo Chu et al. 2018. Joint cache resource allocation and request routing for in-network caching services. *Comp. Net.* 131 (2018), 1–14. arXiv:arXiv:1710.11376v2
[10] A. Di Stefano, A. Di Stefano, G. Morana, and D. Zito. 2018. Coope4M: A Deployment Framework for Communication-Intensive Applications on Mesos. In *2018 IEEE 27th Int. Conf. on Enabling Technol.: Infrastructure for Collaborative Enterprises (WETICE)*. 36–41.
[11] Koustabh Dolui and Soumya Kanti Datta. 2017. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. *2017 Global Internet of Things Summit (GIoTS)* (2017), 1–6.
[12] M. Enguehard, G. Carofiglio, and D. Rossi. 2018. A Popularity-Based Approach for Effective Cloud Offload in Fog Deployments. In *ITC*.
[13] S. Ghosh et al. 2003. Scalable resource allocation for multi-processor QoS optimization. In *IEEE ICDCS*. 174–183.
[14] Yichao Jin, Yonggang Wen, and Cedric Westphal. 2015. Optimal Transcoding and Caching for Adaptive Streaming in Media Cloud. *IEEE Trans. Circ. and Sys. Video Tech.* 25, 12 (2015), 1914–1925.
[15] Sladana Josilo et al. 2019. Wireless and Computing Resource Allocation for SelfishComputation Offloading in Edge Computing. In *IEEE INFOCOM*.
[16] Sladana Josilo and Gyorgy Dan. 2018. Selfish Decentralized Computation Offloading for Mobile Cloud Computing in Dense Wireless Networks. *IEEE Trans. Mobile Comput.* 1233, c (2018).
[17] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack Problems* (1st ed.). Springer.
[18] David G Kirkpatrick and Raimund Seidel. 1986. The ultimate planar convex hull algorithm? *SIAM J. on Comp.* 15, 1 (1986), 287–299.
[19] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. 1999. A scalable solution to the multi-resource QoS problem. In *IEEE Real Time Systems Symp.*
[20] J. Liang et al. 2014. When HTTPS Meets CDN: A Case of Authentication in Delegated Service. In *IEEE Symp. on Security and Privacy*.
[21] Bruce M. Maggs and Ramesh K. Sitaraman. 2015. Algorithmic Nuggets in Content Delivery. *ACM SIGCOMM CCR* 45, 3 (2015), 52–66.
[22] S. Maheshwari et al. 2018. Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications. In *2018 IEEE/ACM Symp. on Edge Computing (SEC)*. 286–299.
[23] Hongzi Mao et al. 2016. Resource Management with Deep Reinforcement Learning. In *ACM Hotnet Workshop*.
[24] J. Monsalve, A. Landwehr, and M. Taufer. 2015. Dynamic CPU Resource Allocation in Containerized Cloud Environments. In *2015 IEEE Int. Conf. on Cluster Computing*. 535–536.
[25] C. Pahl and B. Lee. 2015. Containers and Clusters for Edge Cloud Architectures - a Technology Review. In *IEEE FiCloud2*.
[26] B. P. Rimal et al. 2018. Experimental Testbed for Edge Computing in Fiber-Wireless Broadband Access Networks. *IEEE Com.Mag.* 56, 8 (2018), 160–167.
[27] Y. Song et al. 2008. Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In *IEEE Military Comm. Conf.*
[28] Y. Tao et al. 2017. Dynamic Resource Allocation Algorithm for Container-Based Service Computing. In *IEEE ISADS*.
[29] Liang Tong, Yong Li, and Wei Gao. 2016. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM*.
[30] Doron Zarchy et al. 2015. Capturing resource tradeoffs in fair multi-resource allocation. In *IEEE INFOCOM*.
[31] D. Zhang and pthers. 2017. Container oriented job scheduling using linear programming model. In *ICIM*.