Multi-Objective Scheduling and Resource Allocation of Kubernetes Replicas Across the Compute Continuum

Nicola Di Cicco[‡], Filippo Poltronieri[†], José Santos[*], Mattia Zaccarini[†], Mauro Tortonesi[†], Cesare Stefanelli[†], Filip de Turck[*]
[‡] Department of Electronics, Information, and Bioengineering (DEIB), Politecnico di Milano, Italy
[†] Distributed Systems Research Group, University of Ferrara, Ferrara, Italy
[*] Ghent University - imec, IDLab, Department of Information Technology, Gent, Belgium

***Abstract*—Orchestrating microservice applications deployed on a federation of globally distributed Kubernetes clusters is a challenging and multifaceted optimization problem. It is not only computationally hard, but also requires balancing a delicate trade-off between competing performance metrics, such as latency, deployment cost, and service interruption frequency. Classical approaches in the literature merge multiple objectives into a single one via, e.g., linear combinations. However, in practice, it is complex to express a priori a quantitative preference between heterogeneous objectives, let alone with simple linear combinations. This paper adopts a more comprehensive approach leveraging proper Multi-Objective Optimization (MOO), with the goal of producing multiple solutions from the Pareto Front (PF). Therefore, the orchestrator can inspect a posteriori all possible "optimal" trade-offs and decide on the strategy that best fits their operating requirements. To solve the MOO problem, this paper adopts state-of-the-art Multi-Objective Evolutionary Algorithms and shows their effectiveness in solving the MOO problem. Illustrative results highlight the practical benefits of a MOO formulation, providing several tens of nondominated solutions and evenly covering the objectives' space.**

***Index Terms*—Kubernetes, Resource Allocation, Multi-Objective Optimization, ILP, Evolutionary Algorithms, Compute Continuum**

I. Introduction

Microservices have revolutionized service provisioning in Cloud Computing environments, allowing providers to distribute complex applications in the Compute Continuum (CC) over loosely-coupled containers [1]–[3]. Specifically, multi-cluster scenarios provide ample degrees of freedom for scheduling and resource allocation of microservices, allowing service providers to select among a large portfolio of compute instances with different characteristics in terms of available CPU, RAM, pricing models, and Quality of Service (QoS) metrics, e.g., round-trip latency and guaranteed availability [4].

In this context, resource allocation in heterogeneous compute clusters is well-studied in the literature, e.g., [4]–[7], and can be framed as a generalization of the multidimensional temporal bin packing problem [8]–[11]. Despite the compelling interest, state-of-the-art automated management solutions (e.g. Amazon EKS, Platform 9) still lack the tools to provide cost-efficient deployment subject to Service Level Agreements (SLAs) such as availability and latency.

In this paper, building upon prior work, we put forward proper Multi-Objective Optimization (MOO) for enabling informed a-posteriori decision-making in multi-cluster Kubernetes (K8s) scheduling and resource allocation. We argue that a Multi-Objective (MO) formulation is necessary for this problem since the complex and unpredictable interactions between heterogeneous objective functions make it challenging, if not impossible, to define precise and explicit preferences in advance. In practice, when dealing with multiple competing objectives, it is desirable to present to the decision-maker a set of multiple "optimal" trade-offs, such that well-informed decisions can be made a posteriori [12], [13].

In this regard, in contrast to single-objective optimization, we aim to derive the entire Pareto Front (PF) of solutions, each achieving "optimal" trade-offs between three competing objectives: service latency, service deployment costs, and frequency of service interruption. Specifically, our work improves over the state-of-the-art by jointly considering the following four main aspects: *i)* we simultaneously optimize both scheduling and resource allocation, *ii)* we assume that the decision-maker's preferences between objectives are unknown before optimizing, *iii)* we consider different pricing and latency models between heterogeneous clusters, which we profile from real-world AWS instances, and *iv)* we consider the possibility of splitting multiple microservices over different compute instances during resource allocation. In this context, we formulate scheduling and resource allocation in K8s clusters as a MO-Integer Linear Programming (ILP) problem and leverage metaheuristics to solve it efficiently.

Our main contributions are summarized as follows:

- We derive a novel MO-ILP formulation for scheduling and resource allocation in K8s clusters, to jointly optimize three competing objectives: service latency, total deployment cost, and service reliability. (Section III)
- We develop several ad-hoc metaheuristics for solving our MO-ILP efficiently. Specifically, we derive a custom

N. Di Cicco and F. Poltronieri are co-first authors of this paper.

and efficient variable encoding suitable for state-of-the-art Genetic Algorithms such as NSGA-II and NSGA-III, and we implement a custom Particle Swarm Optimization (PSO) specifically designed for MOO. (Section IV)
- We perform extensive numerical evaluations based on real-world round-trip delay measurements, instance pricing models, and reliability indicators, and we discuss our observed trade-offs by analyzing the PFs. (Section V)

II. Related Work

ILP-based optimization has been extensively applied in a wide range of problems in networking, ranging from microservice orchestration in cloud systems [14] to service chain management [15]. Typically, resource allocation problems are modeled as ILPs, which are typically solved via ad-hoc heuristics for large-scale instances to cope with the scalability issues arising from ILP solvers [16], [17].

Thanks to their flexibility, ILPs have also been considered for modeling MOO problems. For example, the authors in [18] formalize a Mixed-Integer Linear Program (MILP) to improve the acceptance rate and the delay and, at the same time, prevent QoS violations for a Virtual Network Embedding (VNE) problem in multiple cloud environments. In [19], Yao et al. aim to examine the correlation between reliability and costs in Internet-of-Things networks deployed at the Fog level.

Due to the high computational complexity for solving MO-ILPs optimally, heuristic algorithms, especially multi-objective Genetic Algorithms such as NSGA-II, are frequently used to obtain approximate solutions in reasonable computational times, e.g., [20]–[23]. For example, the work in [21] leverages NSGA-II to tackle network reliability and controller load balancing in Software Defined Networks. The authors empirically demonstrate its effectiveness on a collection of different network configurations. In [20] the authors model an adaptation of NSGA-II for discretized objectives in a task scheduling problem. Their experiments prove the strength of the proposed approach, minimizing execution times and costs in the considered fog-cloud environments. In [22], the authors deal with a workflow scheduling problem in cloud ecosystems leveraging a PSO tailored to deal with multiple conflicting objectives, such as makespan and resource usage. In [23], the authors compare reinforcement learning and metaheuristics techniques for service management across the CC. Specifically, the authors analyze the convergence of Computational Intelligence (CI) and Reinforcement Learning (RL) techniques and discuss their adoptions for dynamic scenarios.

A large body of literature is dedicated to scheduling and resource allocation in cloud computing, and ILP-based approaches [5], [6]. The general mathematical framework of the problem is the Temporal Multi-Dimensional Bin Packing Problem [8]–[11], where bins (i.e., compute instances) have multiple dimensions of capacity (e.g., CPU and RAM). Service requests have to be scheduled over the time axis, either by considering a fixed scheduling period or per-request deadlines. The works that share the most similarity with our paper are [4]–[6]. We now detail similarities and differences compared to our work, highlighting our novel contributions.

In [5], the authors consider a multi-objective virtual machine placement problem to jointly minimize the total number of used physical machines and the number of "fire-ups", i.e., the number of times a machine is switched off and back on. In contrast to our paper, this work considers objectives related to the deployment cost of virtual machines. Instead, our model also incorporates QoS metrics (namely, latency and probability of service interruption). Moreover, this work assumes that virtual machines must be integrally allocated to one compute instance, while we admit the possibility of splitting microservices over different compute instances. Furthermore, [5] assumes that all compute nodes are identical. Instead, our work considers heterogeneous compute nodes with different pricing models and QoS metrics. Finally, [5] considers only resource allocation without temporal scheduling.

In [6], the authors consider the problem of resource allocation of long-running microservices applications in shared clusters to minimize the number of utilized hardware nodes. Specifically, the authors admit the possibility of splitting microservices over multiple shared clusters and impose further constraints prohibiting the co-location of incompatible microservices. Our work extends this general formulation by jointly considering multiple objectives besides hardware costs, namely, service latency and availability. Moreover, similarly to [5], this work assumes that compute instances are identical and does not consider temporal scheduling.

In [4], the authors consider online multi-cluster admission control and resource allocation of K8s replicas. Specifically, decisions on whether or not to admit a request and in which cluster to assign a request are taken on the fly by a Reinforcement Learning agent. Instead, in this paper, we consider offline scheduling and resource allocation of multiple requests, a significantly larger-scale problem requiring different solving methods. Moreover, though [4] considers multiple goals (e.g., deployment cost, latency, and fairness), it resorts to manually-tuned objective weighting. Instead, our work considers a principled MO approach, aiming to compute a PF of solutions without the need for specifying the objectives' preferences.

In summary, this paper significantly extends the state-of-the-art by jointly *i)* considering both scheduling and resource allocation, *ii)* considering a MO approach simultaneously optimizing service latency, deployment cost, and microservice availability, *iii)* considering heterogeneous compute clusters with different resource availability and performance metrics, and *iv)* admitting the possibility of splitting microservices over multiple compute clusters.

III. Background on Multi-Objective Optimization

MOO addresses optimization problems with two or more conflicting objective functions to be optimized simultaneously, seeking to find multiple solutions that balance the trade-offs

between the objectives. Specifically, a MO-ILP problem can be generally written as:

$$\min \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})) \tag{1}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{2}$$

$$\mathbf{x} \in \mathbb{Z}^n \tag{3}$$

where $(f_1, f_2, \ldots, f_k)$ are the objective functions, $\mathbf{x}$ are the decision variables, and $\mathbf{A}, \mathbf{b}$ are the constraint matrix and the right-hand-side coefficients, respectively.

In contrast to single-objective optimization, a MO-ILP has in general multiple "optimal" solutions, each achieving a different trade-off between the objectives. Formally, the set of optimal solutions to the MO-ILP consists of its Pareto Front (PF), which is defined as:

$$\mathcal{F} = \{\mathbf{x} \in \Omega \mid \mathbf{f}(\mathbf{x}) \succ_P \mathbf{f}(\mathbf{x}') \ \ \forall \mathbf{x}' \in \Omega\}, \tag{4}$$

where Ω is the feasible set of the MO-ILP, and $\succ_P$ is the Pareto dominance relation, defined as:

$$\mathbf{x} \succ_P \mathbf{x}' \iff (f_i(\mathbf{x}) \geq f_i(\mathbf{x}') \ \forall i) \wedge (\exists i \colon f_i(\mathbf{x}) > f_i(\mathbf{x}')) \tag{5}$$

In other words, a feasible solution $\mathbf{x}$ Pareto-dominates $\mathbf{x}'$ if and only if $\mathbf{x}$ improves over $\mathbf{x}'$ in at least one objective while not worsening the others [24].

Deriving the full PF is significantly more complex than conventional single-objective optimization, as it requires purposely designed solving algorithms. To address the complexity of MOO, a simple approach is to *scalarize* the objectives into a single one via a scalarization function $s \colon \mathbb{R}^k \to \mathbb{R}$. A popular approach for MO-ILP problems is defining s as a convex combination of the objectives, such that the linearity of the objective function is preserved, and thus the problem can be solved as a single-objective ILP with conventional algorithms. According to this approach, each objective is assigned a scalar weight, whose value represents the importance of the specific objective function among the other. However, there are two main drawbacks to this approach. First, it requires specifying the relative importance of the objectives a priori. Since the interactions between the objectives are often nonlinear, predicting the effect of changing a coefficient in the resulting solution is impossible. Second, linear combinations can only achieve solutions lying on the convex hull of the PF, which can result in missing many potentially interesting trade-offs. For the above reason, proper MOO algorithms aim to either enumerate or approximate the PF, such that the decision-maker is presented with multiple solutions, and is then able to decide a posteriori on the trade-off that best satisfies the system's operational requirements.

IV. System Model

We consider a scenario such as the one illustrated in Fig. 1, where a service provider must schedule and assign compute resources to a set of service requests over the Compute Continuum (CC), considering multiple DCs and instance types. Specifically, we assume that requests consist of multiple microservice replicas, which can possibly be distributed over

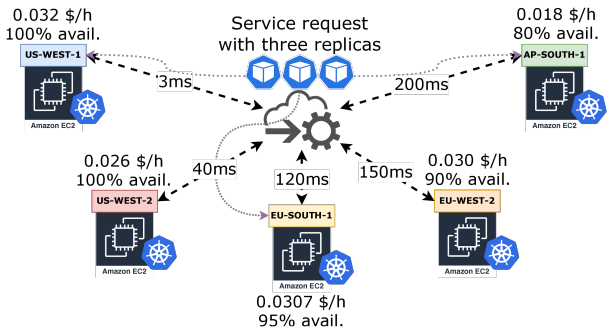


Fig. 1. Orchestration of a service request in a multi-cluster CC scenario. Each request is associated with a geographical area corresponding to the AWS DC locations. Scheduling and resource allocation decisions are influenced by the latency and pricing of the compute instances. (Numbers are illustrative.)

TABLE I
PARAMETERS OF THE MO-ILP

Parameter	Description
D	Set of service requests
D_t	Set of service requests that are **active** at time-slot t
DC	Set of data centers
I	Set of compute instances available in each data center
$I_r \subset I$	Set of RESERVED instances available in each DC
$I_o \subset I$	Set of ON-DEMAND instances available in each DC
$I_s \subset I$	Set of SPOT instances available in each DC
$P^r_{j,k}$	Total price for RESERVED instance $j \in I_r$ in data center $k \in$ DC instance over the complete time horizon
$P^o_{j,k}$	Hourly price for ON-DEMAND instance $j \in I_o$ in data center $k \in$ DC
$P^s_{j,k}$	Hourly price for SPOT instance $j \in I_s$ in data center $k \in$ DC
R_i	Set of replicas required by request $i \in D$
T	Set of time-slots over the scheduling period
c_i	CPU required by one replica of request $i \in D$
m_i	RAM required by one replica of request $i \in D$
C_j	vCPUs offered by instance $j \in I$
M_j	RAM offered by instance $j \in I$
$f_{j,k}$	Interruption frequency of SPOT instance $j \in I_s$ in data center $k \in$ DC
$l_{i,k}$	Latency between request $i \in D$ and data center k

different K8s clusters in the CC and coordinated via a global topology manager [4].

Formally, we model this optimization problem as a MO-ILP. The MO-ILP is given as an input *i)* a set of deployment requests, characterized by CPU and RAM requirements, a request duration, and a required number of microservice replicas, and *ii)* a set of potential compute resources, representing RESERVED, ON-DEMAND or SPOT cloud instances, each characterized by a specific data center location, pricing model, hardware resource availability, and average frequency of service interruption. The objective of the ILP is to decide on a scheduling and admission control rule that jointly optimizes the acceptance rate, the hardware costs, and the average reliability. Table I and Table II illustrate the parameters and decision variables, respectively, of the MO-ILP formulation.

First, we describe in detail our considered objective functions. Our first goal is to minimize the maximum delay experienced by each request, which we express as follows:

$$\min f_1 = \min \frac{1}{|D|} \sum_{i \in D} L_i. \tag{6}$$

TABLE II
DECISION VARIABLES OF THE MO-ILP

Variables	Description
$r_{j,k}$	1 if RESERVED instance $j \in I_r$ in data center $k \in$ DC is active, 0 otherwise.
$o^t_{j,k}$	1 if ON-DEMAND instance $j \in I_o$ is active at time-slot $t \in T$ in data center $k \in$ DC, 0 otherwise
$s^t_{j,k}$	1 if SPOT instance $j \in I_s$ is active at time-slot $t \in T$ in data center $k \in$ DC, 0 otherwise
a^t_i	1 if request $i \in D$ is scheduled to start at time-slot $t \in T$, 0 otherwise.
$x^{t'}_{i,j,k,r}$	1 if K8s replica r of request $i \in D$ is scheduled to start at time-slot $t' \in T$ and is allocated to the instance $j \in I$ in data center $k \in$ DC, 0 otherwise.
$d_{i,k}$	1 if request $i \in D$ is assigned to data center $k \in$ DC, 0 otherwise
L_i	Maximum latency experienced by request $i \in D$

Our second goal is to minimize the total cost of leasing the hardware instances. Specifically, we adopt a realistic pricing model inspired by AWS EC2 [25], [26], considering three types of cloud computing instances, namely, RESERVED, ON-DEMAND, and SPOT. RESERVED instances offer a low hourly rate, but require reserving capacity for one or multiple years. For this reason, we assume that, in the case a RESERVED instance is activated, its cost is fixed respectively from usage, and amounts to the total cost sustained over the considered scheduling time period. ON-DEMAND instances offer a higher hourly rate than RESERVED instances, but can be instantiated on the fly (e.g., to sustain a small amount of demands, which would not justify instantiating a RESERVED instance), and are paid only for the amount of time they are used. Finally, SPOT instances are the cheapest instances available. These offer unused capacity on the cloud at a highly discounted rate compared to RESERVED and ON-DEMAND instances, can be instantiated on the fly, and are paid by their usage. However, they offer a low level of reliability, since the allocated capacity is not guaranteed, but can be requested back by the cloud provider. By considering these instance types and their pricing models, we express cost minimization as follows:

$$\min f_2 = \min \sum_{k\in \text{DC}} \sum_{j\in I_r} P^r_{j,k} r_{j,k} + \sum_{k\in \text{DC}} \sum_{t\in T} \sum_{j\in I_s} P^o_{j,k} o^t_{j,k} + \sum_{k\in \text{DC}} \sum_{t\in T} \sum_{j\in I_s} P^s_{j,k} s^t_{j,k} \tag{7}$$

Note that cost minimization competes with latency minimization. In general, the cheapest cluster instances are not necessarily the ones offering the lowest latency to a request. For example, with reference to the parameters illustrated in Tables III and IV, consider a set of requests coming from the EU-SOUTH-1 area. Suppose that we assign all requests to ON-DEMAND instances. Deploying them in AP-SOUTH-1 instead of the closer EU-SOUTH-1 instances would decrease the hourly rate for a single ON-DEMAND instance by 1.7x, but would increase the average latency by 43x. In contrast, assigning requests to EU-SOUTH-1 Spot instances would instead reduce the hourly rate by 4x, at the price of a 10% average frequency of disruption. Thus, we ask: *Which solution is best? Are there any preferable middle-ground solutions between these extremes?* Answering these questions before solving the optimization problem requires quantifying the relative preferences between the two objectives and formalizing them into a single-objective formulation. Instead, adopting a MO formulation allows us to derive a set of "locally optimal" trade-offs, and decide which to deploy a posteriori.

Finally, our third objective minimizes the average frequency of replica interruption. Recall that Spot instances are the cheapest compute instances available, but their offered capacity can be subject to interruptions. Thus, if one or more K8s replicas are allocated to Spot instances, these might experience a service interruption if the cloud provider must request back the capacity to, e.g., allocate more ON-DEMAND or RESERVED instances for other clients. For this reason, we want to minimize the average interruption frequency relative to the total number of allocated replicas, as follows:

$$\min f_3 = \min \frac{1}{|D|} \sum_{i\in D} \sum_{j\in I_s} \sum_{k\in \text{DC}} \sum_{r\in R_i} \sum_{t\in T} \frac{z^t_{i,j,k,r} f_{j,k}}{r_i} \tag{8}$$

With arguments similar to our previous discussion, we can see from Table III and Table IV that minimizing the average interruption frequency is in competition with both latency and cost minimization. In other words, the Spot instances offering the lower interruption frequency are not necessarily the ones offering both the lowest latency and costs.

By jointly considering these three objective functions, we formulate our optimization problem as a MO-ILP. Specifically, we pre-compute the set D_t for each time-slot, containing the indexes of all scheduling and resource allocation assignments (i.e., starting time-slot, instance, and DC assignment) that are "active" (i.e., they must occupy CPU and RAM resources) at time-slot t. The MO-ILP formulation, comprising all the necessary constraints, can be expressed as follows:

$$\min (f_1, f_2, f_3) \tag{9}$$

$$\sum_{\substack{(i,t')\in D_t,\\ r\in R_i}} x^{t'}_{i,j,k,r} c_i \le \begin{cases} r_{j,k} C_j, & \forall j \in I_r, k \in \text{DC}, t \in T \\ o^t_{j,k} C_j, & \forall j \in I_o, k \in \text{DC}, t \in T \\ s^t_{j,k} C_j, & \forall j \in I_s, k \in \text{DC}, t \in T \end{cases} \tag{10}$$

$$\sum_{\substack{(i,t')\in D_t,\\ r\in R_i}} x^{t'}_{i,j,k,r} m_i \le \begin{cases} r_{j,k} M_j, & \forall j \in I_r, k \in \text{DC}, t \in T \\ o^t_{j,k} M_j, & \forall j \in I_o, k \in \text{DC}, t \in T \\ s^t_{j,k} M_j, & \forall j \in I_s, k \in \text{DC}, t \in T \end{cases} \tag{11}$$

$$\sum_{j\in I} \sum_{k\in \text{DC}} \sum_{t\in T} x^t_{i,j,k,r} = 1, \qquad \forall i \in D, r \in R_i \tag{12}$$

$$x^t_{i,j,k,r} \le a^t_i, \quad \forall i \in D, j \in I, k \in \text{DC}, r \in R_i, t \in T \tag{13}$$

$$\sum_{t\in T} a^t_i = 1, \qquad \forall i \in D \tag{14}$$

$$d_{i,k} \ge \sum_{t\in T, j\in I} x^t_{i,j,k,r}, \qquad \forall i \in D, k \in \text{DC}, r \in R_i \tag{15}$$

$$L_i \ge d_{i,k} l_{i,k}, \qquad \forall i \in D, k \in \text{DC} \tag{16}$$

$$r_{j,k}, o^t_{j,k}, s^t_{j,k}, a^t_i, x^t_{i,j,k,r}, d_{i,k} \in \{0, 1\} \tag{17}$$

$$L_i \geq 0 \tag{18}$$

Constraints (10) and (11) impose the CPU and RAM constraints, respectively, for RESERVED, ON-DEMAND, and SPOT instances. Constraints (12) impose that each request must be allocated its demanded amount of replicas, which can distributed across every available instance type and DC. Note that, while multiple replicas can be split across multiple instances and DCs, a single replica must be integrally allocated to a single instance and a single DC. Constraints (13) and (14) ensure that only one starting time can be chosen for scheduling each service request. Constraints (15) and (16) set the maximum delay experienced by each allocated request. Constraints (17) and (18) impose the variable domains. Fig. 2 shows an illustrative feasible solution for this problem.

This problem, along with similar problems in the context of packing and scheduling, is NP-Hard. Specifically, our problem can be considered a variant of the Temporal Multidimensional Bin Packing Problem, a popular ILP model for representing resource allocation problems in cloud computing environments [8]–[11]. In our problem, differently from the literature, the bins (i.e., the compute clusters) are not assumed to be identical and have different capacities, pricing models, network latencies, and frequencies of service interruption depending on the bin type. All of these performance metrics play a role in their respective objective function.

The optimal solution to this MO-ILP is the complete Pareto Front, i.e., in a set of solutions covering all possible "optimal" trade-offs between the three objectives. Enumerating the Pareto Front is computationally challenging for large-scale instances, since it would require solving to optimality hundreds, or even thousands of single-objective ILPs [27]. We now discuss several exact and heuristic computational strategies for efficiently computing Pareto-optimal solutions.

A. ILP-based Multi-Objective (MO) Algorithms

A popular and effective approach for solving MO-ILP problems is leveraging conventional single-objective ILP solvers to either enumerate or approximate the Pareto Front [28]–[30]. Intuitively, the idea is to solve a sequence of suitable single-objective ILP problems, such that each solver call returns a Pareto-optimal solution. A classical and simple example of these strategies is the ε-constraint method [28], which optimizes one objective at a time while constraining all other objectives to be less or equal to a constant value ε. The algorithm can return a good approximation of the Pareto Front by choosing suitable discretization strategies for the objectives. Modern exact and approximating algorithm, e.g., [29], [30], refine this general methodology to further reduce the number of solver calls. Unfortunately, small-scale instances (e.g., 10 service requests with 1-2 replicas per request) of a single-objective version of our problem require hundreds of seconds to be solved to optimality by state-of-the-art open-source solvers such as HiGHS [31]. For this reason, we resort to scalable metaheuristics to compute approximate Pareto Fronts for our problem, which we describe as follows.

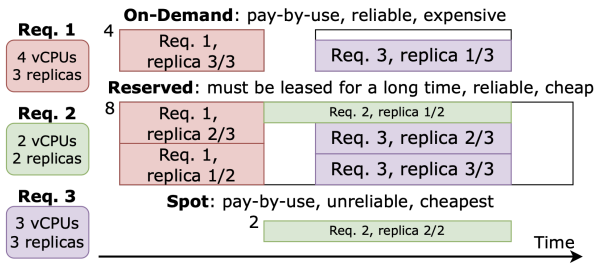


Fig. 2. Illustrative feasible solution MO scheduling and resource allocation of K8s replicas, considering only vCPUs for ease of visualization. Replicas may be split across multiple compute instances, each with different characteristics, and must start at the same time-slot.

B. Multi-Objective Evolutionary Algorithms (EAs)

Thanks to their scalability, Evolutionary Algorithms (EAs), are a popular and effective solution for approximately solving MO problems. These algorithms combine conventional crossover and mutation operators of classical EAs with ad-hoc heuristics for maximizing the diversity of the discovered Pareto Front [32]–[34]. We now briefly outline some of the most popular and effective algorithms in the literature, which we also employ in our work.

NSGA-II [33]: Born as an improved version of NSGA [35], it is characterized by three core design elements: i) a fast sorting procedure for identifying Pareto-dominating solutions in short computational times, ii) the introduction of a crowding distance metric to maximize the spread of the solutions over the Pareto Front, and iii) an elitist approach, which preserves the best-found Pareto-dominating solutions at each iteration. The general applicability and effectiveness of NSGA-II make it a de-facto default choice for MO problems.

NSGA-III [36]: improves over [33] for MO problems with a large number of objectives. To maximize the approximated Pareto Front's diversity, instead of computing a crowding distance as in NSGA-II, it assigns each solution to a reference point in a normalized objective space, which the algorithm tries to cover uniformly. For high dimensions (i.e., 3 or more objectives), this procedure is more effective and computationally lighter than the crowding distance.

Particle Swarm Optimization (PSO) [34]: In addition to the characteristics of its architecture as an evolutionary algorithm, PSO is particularly suitable for multi-objective optimization thanks to its notable convergence speed and efficient exploration of the solution space. The standard approach for MO with PSO algorithms relies on an external repository to store non-dominated feasible solutions and use them to guide the evolution process. Over the years, many adaptations have been made to its structure, such as the selection methodologies to find local and global bests, the mutation operators [37], and the number of total swarms involved [38], [39]. In particular, the last aspect demonstrated to be particularly promising thanks to its ability to explore the solution space efficiently and the possibility of delegating every individual objective of the problem to a specific swarm. For this reason, we devel-

oped a custom **Multi-Swarm Particle Swarm Optimization (MSPSO)** inspired by the work described in [39]. Specifically, our implementation creates a swarm with a fixed number of particles for each objective, and adopts strategies to influence the movement of the particles (i.e., the exploration of the solution space) tailored for dealing with integer solutions, such as Integer Polynomial mutation and a custom comparator to select dominated solutions while taking into account penalties related to constraint violations. We adopt this custom operators also for our NSGA-II and NSGA-III implementation.[1]

V. Illustrative Numerical Results

In this Section, we solve the MO-ILP described in Section IV and analyze the resulting PF of solutions. For visualization purposes, we first solve a restricted version of our MO problem considering only two competing objectives at a time, namely, latency with cost and cost with replica interruption frequency. Then, we solve the complete MO problem considering all three objectives simultaneously. We solve the problem via NSGA-II, NSGA-III, and MSPSO (implemented via jMetalPy [41]).

A. Reference scenario

We consider a large-scale instance similar in size to those reported in the literature [7], [42], consisting of 50 requests with replicas ranging from 1 to 6 and a duration ranging from 10 to 50 time-slots (hours), over a scheduling period of 100 time-slots (hours).[2] Each request is characterized by a per-replica required number of vCPUs (number of virtual cores) and RAM, a duration, i.e., the time the deployed replicas must remain active, and a requested number of replicas. To model the round-trip latency, we assume that requests originate from a location close to one of the AWS EC2 DCs.

To realistically model the cloud instance specifications alongside their pricing models, we select the *t4g AWS EC2* computing instances for the configurations 2XLARGE, LARGE, and SMALL to specify different capacities. For each one of these computing instances, we collected the hardware specifications and pricing details for the respective On-Demand, Reserved, and Spot renting options, which are illustrated in Table III. Specifically, we consider six EC2 DCs, located in different geographical regions: EU-SOUTH-1, EU-WEST-2, EU-EAST-1, US-WEST-1, AP-NORTHEAST-3, and AP-SOUTH-1. Furthermore, for Spot instances, we also report the interrupt frequency, which is the maximum frequency at which instances' interruption occurred according to Amazon's official reports.[3] Finally, we modeled the latency between the different EC2 DCs according to the Cloud Ping website [40], from which we retrieved the average latency, i.e., the ping, collected over a one-month period between each pair of EC2 DC considered for this evaluation.

[1]The source code for this study is available at https://github.com/DSG-UniFE/cnsm_2024_moo_scheduling_ra.

[2]As a rough estimate of the problem's scale, considering six DCs and three instance types per DC, such an instance admits millions of possible time-slots and instance assignments.

[3]https://aws.amazon.com/it/ec2/spot/instance-advisor/

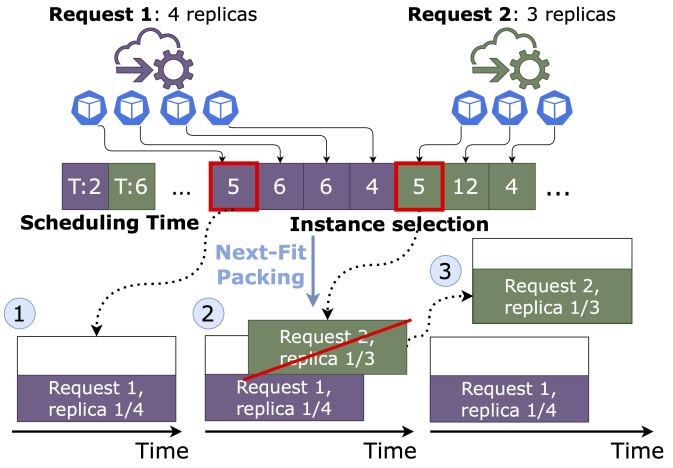


Fig. 3. Scheduling and RA decisions taken with MOEA. The encoding vector is divided into two parts: the first decides the scheduling for each request, and the second assigns each replica to a proper instance type (DC, type, size), encoded as an integer. Then, instances are created according to a next-fit algorithm: replicas are packed sequentially, and new instances are opened as soon as the capacity is exceeded.

To numerically assess the quality of the approximated PFs, we consider the following two performance metrics.

Hypervolume metric. Measures the volume of the PFs relative to a fixed reference point in the objective space, e.g., the worst possible values that the objective functions can take. A higher hypervolume metric signals a better approximation of the true Pareto Front.

Sparsity. Measures the resolution of the approximated PF. Intuitively, a higher resolution (i.e., a large number of solutions in the PF) is more desirable, since it provides a greater degree of flexibility to the orchestrator. Sparsity is defined as [43]:

$$S(\mathcal{F}) = \frac{1}{|\mathcal{F}-1|} \sum_{j=1}^{m} \sum_{i=1}^{|\mathcal{F}-1|} \left(\tilde{\mathcal{F}}_j(i) - \tilde{\mathcal{F}}_j(i+1) \right)^2 \qquad (19)$$

where m is the number of objectives, $\mathcal{F}$ is the PF approximation, and $\tilde{\mathcal{F}}_j(i)$ is the value of the j-th objective of the i-th solution, sorted by objective value. Lower sparsity implies a higher resolution of the approximated PF.

Before presenting the results, we detail our custom solution encoding for solving the MO problem with Multi-Objective Evolutionary Algorithms (MOEA)s.

B. MOEA Encoding

MOEAs require encoding a feasible solution of the optimization problem in a vector representation. Then, operators such as mutation as crossover are iteratively applied to a population of multiple vector representations to explore efficiently the solution space [33], [36].

To encode the scheduling and resource allocation problem, we design an integer vectorized representation illustrated in Fig. 3, which shows both the scheduling and the resource allocation phases. The encoding vector is divided into two main parts: the first contains one element for each request to map the respective scheduling decisions, i.e., when to schedule

TABLE III
HARDWARE CONFIGURATION OF EACH CLUSTER BASED ON AMAZON EC2 ON-DEMAND AND SPOT PRICING [25], [26].

Data Center (DC)	Instance	On-Demand ($/h)	Reserved On-Demand ($/h)	Spot ($/h)	vCPU	RAM	Spot Interrupt Frequency
eu-south-1 (Milan)	t4g.2xlarge	0.3072	0.1842	0.0308	8	32.0	10%
	t4g.large	0.0768	0.0461	0.078	2	8.0	15%
	t4g.small	0.0192	0.0115	0.0019	2	2.0	10%
eu-west-2 (London)	t4g.2xlarge	0.3008	0.1808	0.01148	8	32.0	5%
	t4g.large	0.0752	0.0452	0.0267	2	8.0	10%
	t4g.small	0.0188	0.0112	0.0061	2	2.0	5%
us-east-1 (N. Virginia)	t4g.2xlarge	0.02688	0.1606	0.1135	8	32.0	15%
	t4g.large	0.0672	0.0402	0.0268	2	8.0	20%
	t4g.small	0.0168	0.0100	0.0090	2	2.0	5%
us-west-1 (N. California)	t4g.2xlarge	0.3200	0.1918	0.0849	8	32.0	10%
	t4g.large	0.0800	0.0480	0.0248	2	8.0	10%
	t4g.small	0.0200	0.0120	0.0061	2	2.0	5%
ap-northeast-3 (Osaka)	t4g.2xlarge	0.3482	0.2089	0.0415	8	32.0	10%
	t4g.large	0.0879	0.0522	0.0298	2	8.0	15%
	t4g.small	0.0218	0.0130	0.0031	2	2.0	15%
ap-south-1 (Mumbai)	t4g.2xlarge	0.1792	0.1080	0.0760	8	32.0	10%
	t4g.large	0.0448	0.0270	0.0211	2	8.0	15%
	t4g.small	0.0112	0.0068	0.0049	2	2.0	5%

TABLE IV
AVERAGE LATENCY (MS) BETWEEN AWS EC2 DCS [40].

DC	eu-s1	eu-w2	us-e1	us-w1	ap-n3	ap-s1
eu-s1	2.57	27.69	102.16	162.51	231.99	110.53
eu-w2	26.92	3.88	77.57	147.85	217.17	119.41
us-e1	102.04	78.57	5.15	64.21	154.10	195.66
us-w1	162.10	147.91	63.61	3.24	109.47	231.06
ap-n3	232.13	217.81	153.88	110.11	2.32	131.98
ap-s1	109.98	119.64	193.90	231.79	131.34	3.23

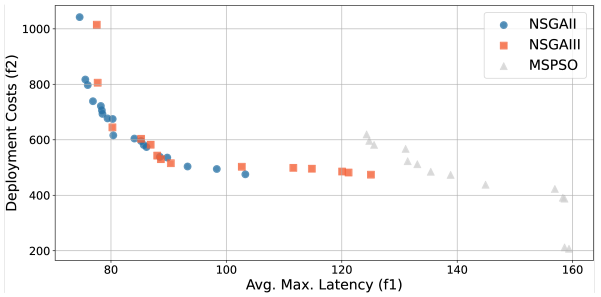
the request in the time horizon, while the second part contains one element for each replica. Specifically, the integer value of the element encodes a specific combination of the EC2 DC, the instance size (small, large, or 2xlarge), and the instance type: ON-DEMAND, RESERVED, or SPOT instance.

We perform decoding (i.e., converting the encoded vector into a feasible solution) according to a classical Next-Fit Bin Packing algorithm, as illustrated in Fig. 3. Specifically, we can identify three different possibilities: i) if there are no active instances for a specific combination, a new instance of the selected type is activated, and the replica is allocated in it; ii) if there exists a running instance for the selected combination, and it has enough CPU and RAM capacity to accommodate the replica, and iii) if there exists a running instance of the selected combination, but it has not enough resources to accommodate the new replica, a new instance of the same time is activated and the replica is allocated in it.

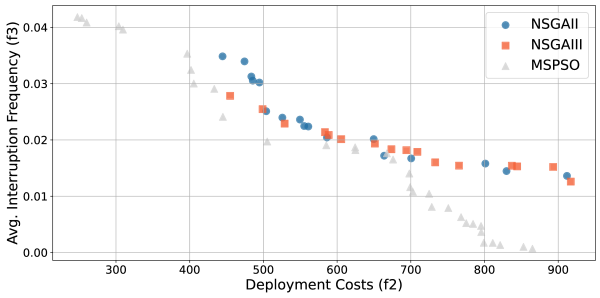
We ran all the experiments on a commodity laptop (MacBook Pro M3) with 50.000 iterations as a termination condition, after which we observed no significant improvements in solution quality. On average, across 100 instances, the experiments took 24 seconds for NSGA-II and NSGA-III and about 110 seconds for MSPSO.

C. MOEA evaluation with two objectives

Fig. 4a and 4b illustrate the PF for bi-objective versions of our problem. We first observe that, when optimizing latency and cost (Fig. 4a), the NSGA algorithms and the



(a)



(b)

Fig. 4. PFs found by NSGA-II, NSGA-III, and MSPSO considering the Deployment Costs and Latency as problem objectives (4a) and Deployment Costs and the Avg. Interruption Frequency as problem objectives (4b).

MSPSO converge on completely different solutions in the objective space. Specifically, while NSGA-II and NSGA-III favor solutions with higher costs and lower latencies, MSPSO favors the opposite. Still, NSGA-II and NSGA-III obtain more practically useful solutions, in correspondence with the "elbow" of the PF, which signals a point of diminishing returns for either objective. In particular, NSGA-II provides the "best" PF approximation since it densely covers the elbow point with multiple solutions, providing the orchestrator with a fine decision-making granularity. In contrast, when optimizing cost and availability (Fig. 4b, the MSPSO produces a PF

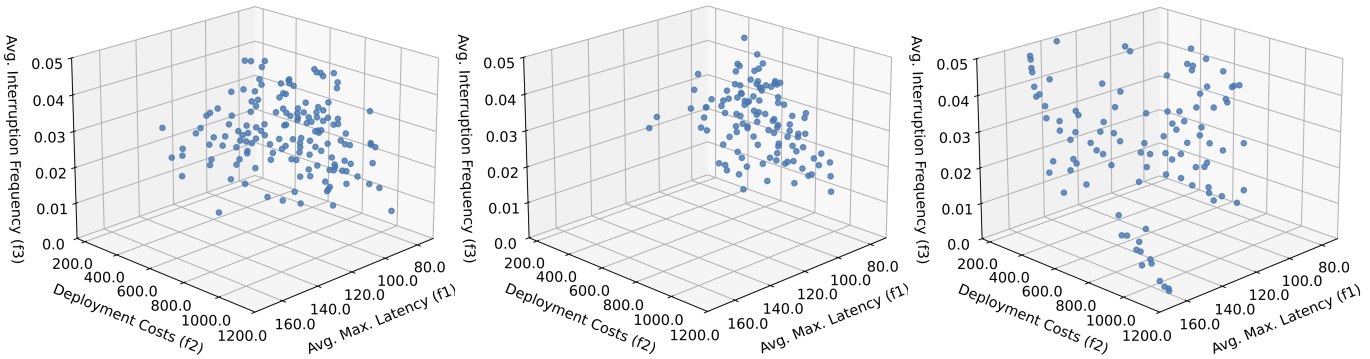


Fig. 5. PFs found by NSGA-II, NSGA-III, and MSPSO, considering total deployment costs, avg. request maximum latency and avg. interruption frequency.

that strongly dominates NSGA-II and NSGA-III, providing a well-spaced set of nondominated solutions with no discernible elbow point signaling diminishing returns for either objective.

We numerically validate the above qualitative comparisons with the quantitative performance metrics illustrated in Table V. Since the algorithms run relatively fast and might focus on completely different parts of the solution space, we recommend running them as an ensemble and combining the resulting PFs. Overall, these results highlight the benefits of a MO approach, producing tens of possible solutions (instead of just one), and providing an explicit way to identify the "ideal" trade-off regions, such as the elbow point in Fig. 4a.

D. MOEA evaluation with three objectives

Fig. 5 illustrates the approximated PFs for the complete three-objective problem. We first observe that the PFs for each algorithm are quite rich, comprising several tens of solutions (150 for NSGA-II, 112 for NSGA-III, and 100 for MSPSO). This is because the flexibility provided by orchestrating single microservices enables a fine decision-making granularity, i.e., the balance between the three objectives can be slightly tilted by moving one or a few replicas from one cluster to another. From a practical perspective, this allows MO algorithms to produce a large set of nondominated solutions, providing ample choices to the orchestrator. Though harder to discern visually compared to the 2D case, the quantitative performance metrics in Table V show that NSGA-III is the best-performing algorithm both in terms of solution quality (measured by the HyperVolume metric) and resolution of the PF (measured by the Sparsity metric), with MSPSO coming second. Indeed, similarly to what we observed in the bi-objective case, we can see how NSGA-III and MSPSO attained convergence in different sections of the solution space. Specifically, NSGA-III discovers solutions with a few percent interruption frequency while trading off cost and latency, while MSPSO discovers solutions with a low interruption frequency but higher costs. As before, considering this complex and multifaceted objective function landscape and the different exploration strategies of metaheuristics, we recommend running all of these algorithms as an ensemble to maximize the quality of the resulting PF and the number of available nondominated solutions.

TABLE V
SUMMARY OF THE MO PERFORMANCE METRICS

Objectives	HV	$S(\mathcal{F})$
Avg. Max. Latency (f1) & Deployment Costs (f2)	NSGA-II : **64264** NSGA-III: 62477 MSPSO: 38293	NSGA-II : 3452 NSGA-III: 5730 MSPSO: **2893**
Deployment Costs (f2) & Avg. Interruption Frequency (f3)	NSGA-II : 15.18 NSGA-III: 15.34 MSPSO: **21.47**	NSGA-II : 1634 NSGA-III: 1276 MSPSO: **887**
Avg. Max. Latency (f1) & Deployment Costs (f2) & Avg. Interruption Frequency (f3)	NSGA-II : 3694 NSGA-III: **4048** MSPSO: 3930	NSGA-II : 203.29 NSGA-III: **123.79** MSPSO: 166.77

VI. CONCLUSION

Scheduling and resource allocation in microservice replicas across the Compute Continuum is a fundamental and challenging optimization problem. In this paper, we proposed a novel MO formulation simultaneously accounting for instance pricing, latency, and interruption frequency. Conventional methods, which often combine these objectives into a single linear equation, fail to capture the complex nature of the trade-offs involved. Instead, we solved the MO problem by computing an approximated PF via custom metaheuristics. This allows decision-makers, such as network managers, to evaluate the spectrum of trade-offs and select the most appropriate strategy for their specific operational needs. We empirically show that, thanks to the flexibility provided by orchestrating individual replicas, the resulting PFs provide several tens of potential solutions (instead of just one, as with conventional approaches), of which we quantify the relative performance and qualitatively identify the trade-off regions that can be of major interest. Our custom metaheuristics are computationally fast for large-scale instances, and converge in distinct regions of the objective space, highlighting the benefit of running them as an ensemble and then combining the resulting PFs. Future work will consider strengthening the MO-ILP formulation (e.g., by the means of covering and column generation approaches), considering production workloads, and validating the solutions' performance on a Digital Twin to perform what-if analyses of multiple Pareto-optimal service deployments [44].

ACKNOWLEDGMENT

This work has been partially supported by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU). José Santos is funded by the Research Foundation Flanders (FWO), grant number 1299323N.

REFERENCES

[1] X. Larrucea *et al.*, "Microservices," *IEEE Software*, vol. 35, no. 3, pp. 96–100, 2018.

[2] N. Dragoni *et al.*, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

[3] J. Santos *et al.*, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[4] ——, "Efficient microservice deployment in kubernetes multi-clusters through reinforcement learning," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–9.

[5] N. Aydın *et al.*, "Multi-objective temporal bin packing problem: An application in cloud computing," *Computers & Operations Research*, vol. 121, p. 104959, 2020.

[6] C. Mommessin *et al.*, "Affinity-aware resource provisioning for long-running applications in shared clusters," *Journal of Parallel and Distributed Computing*, vol. 177, pp. 1–16, 2023.

[7] Y. Hu *et al.*, "Multi-objective container deployment on heterogeneous clusters," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2019, pp. 592–599.

[8] E. M. Arkin *et al.*, "Scheduling jobs with fixed start and end times," *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1–8, 1987.

[9] A. Lodi *et al.*, "Two-dimensional packing problems: A survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002.

[10] M. Dell'Amico *et al.*, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, p. 104825, 2020.

[11] J. Martinovic *et al.*, "Compact integer linear programming formulations for the temporal bin packing problem with fire-ups," *Computers & Operations Research*, vol. 132, p. 105288, 2021.

[12] R. Marler *et al.*, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, 04 2004.

[13] C. F. Hayes *et al.*, "A practical guide to multi-objective reinforcement learning and planning," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 1, p. 26, 2022.

[14] M. Zambianco *et al.*, "Cost minimization in multi-cloud systems with runtime microservice re-orchestration," in *2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN)*, 2024, pp. 65–72.

[15] J. Santos *et al.*, "Efficient orchestration of service chains in fog computing for immersive media," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 139–145.

[16] S. Long *et al.*, "A global cost-aware container scheduling strategy in cloud data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2752–2766, 2022.

[17] L. N. Vijouyeh *et al.*, "Efficient application deployment in fog-enabled infrastructures," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–9.

[18] M. Diallo *et al.*, "A qos-based splitting strategy for a resource embedding across multiple cloud providers," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1278–1291, 2021.

[19] J. Yao *et al.*, "Fog resource provisioning in reliability-aware iot networks," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8262–8269, 2019.

[20] I. M. Ali *et al.*, "An automated task scheduling model using non-dominated sorting genetic algorithm ii for fog-cloud systems," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2294–2308, 2022.

[21] J.-M. Sanner *et al.*, "An evolutionary controllers' placement algorithm for reliable sdn networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–6.

[22] Shubham *et al.*, "An effective multi-objective workflow scheduling in cloud computing: A pso based approach," in *2016 Ninth International Conference on Contemporary Computing (IC3)*, 2016, pp. 1–6.

[23] F. Poltronieri *et al.*, "Reinforcement learning vs. computational intelligence: Comparing service management approaches for the cloud continuum," *Future Internet*, vol. 15, no. 11, 2023.

[24] T. Tušar *et al.*, "Visualization of pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 225–245, 2014.

[25] Amazon AWS, "Amazon ec2 on-demand pricing." accessed on 1 July 2024. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[26] ——, "Amazon ec2 reserved instance pricing." accessed on 1 July 2024. [Online]. Available: https://aws.amazon.com/ec2/pricing/reserved-instances/pricing/?nc1=h_ls.

[27] J. Legriel *et al.*, "Approximating the pareto front of multi-criteria optimization problems," in *Tools and Algorithms for the Construction and Analysis of Systems*, J. Esparza *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 69–83.

[28] G. Mavrotas, "Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems," *Applied Mathematics and Computation*, vol. 213, no. 2, pp. 455–465, 2009.

[29] S. Tamby *et al.*, "Enumeration of the nondominated set of multiobjective discrete optimization problems," *INFORMS Journal on Computing*, vol. 33, no. 1, pp. 72–85, 2021.

[30] M. Ángel Domínguez-Ríos *et al.*, "Effective anytime algorithm for multiobjective combinatorial optimization problems," *Information Sciences*, vol. 565, pp. 210–228, 2021.

[31] Q. Huangfu *et al.*, "Parallelizing the dual revised simplex method," *Mathematical Programming Computation*, vol. 10, no. 1, pp. 119–142, 2018.

[32] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*. L. Erlbaum Associates Inc., 1985, p. 93–100.

[33] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[34] C. Coello *et al.*, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.

[35] N. Srinivas *et al.*, "Multiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[36] J. Blank *et al.*, "Investigating the normalization procedure of nsga-iii," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2019, pp. 229–240.

[37] M. R. Sierra *et al.*, "Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance," in *Evolutionary Multi-Criterion Optimization*, C. A. Coello Coello *et al.*, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 505–519.

[38] S. Sedarous *et al.*, "Multi-swarm multi-objective optimization based on a hybrid strategy," *Alexandria Engineering Journal*, vol. 57, no. 3, pp. 1619–1629, 2018.

[39] Z.-H. Zhan *et al.*, "Multiple populations for multiple objectives: A coevolutionary technique for solving multiobjective optimization problems," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 445–463, 2013.

[40] CloudPing, "Aws latency monitoring." accessed on 1 July 2024. [Online]. Available: https://www.cloudping.co/grid/latency/timeframe/1M.

[41] A. Benítez-Hidalgo *et al.*, "jmetalpy: A python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, vol. 51, p. 100598, 2019.

[42] V. Bracke *et al.*, "A multiobjective metaheuristic-based container consolidation model for cloud application performance improvement," *Journal of Network and Systems Management*, vol. 32, no. 3, p. 61, 2024.

[43] J. Xu *et al.*, "Prediction-guided multi-objective reinforcement learning for continuous robot control," in *International conference on machine learning*. PMLR, 2020, pp. 10 607–10 616.

[44] D. Borsatti *et al.*, "Kubetwin: A digital twin framework for kubernetes deployments at scale," *IEEE Transactions on Network and Service Management*, 2024.