SPACE4AI-R: a Runtime Management Tool for AI Applications Component Placement and Resource Scaling in Computing Continua

Federica Filippini
federica.filippini@polimi.it
Politecnico di Milano
Milano, Italy

Hamta Sedghani
hamta.sedghani@polimi.it
Politecnico di Milano
Milano, Italy

Danilo Ardagna
danilo.ardagna@polimi.it
Politecnico di Milano
Milano, Italy

Abstract

The recent migration towards Internet of Things determined the rise of a Computing Continuum paradigm where Edge and Cloud resources coordinate to support the execution of Artificial Intelligence (AI) applications, becoming the foundation of use-cases spanning from predictive maintenance to machine vision and healthcare. This generates a fragmented scenario where computing and storage power are distributed among multiple devices with highly heterogeneous capacities. The runtime management of AI applications executed in the Computing Continuum is challenging, and requires ad-hoc solutions. We propose SPACE4AI-R, which combines Random Search and Stochastic Local Search algorithms to cope with workload fluctuations by identifying the minimum-cost reconfiguration of the initial production deployment, while providing performance guarantees across heterogeneous resources including Edge devices and servers, Cloud GPU-based Virtual Machines and Function as a Service solutions. Experimental results prove the efficacy of our tool, yielding up to 60% cost reductions against a static design-time placement, with a maximum execution time under 1.5*s* in the most complex scenarios.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Computer systems organization** → **Distributed architectures**; • **Mathematics of computing** → **Optimization with randomized search heuristics**.

Keywords: Component placement, Edge computing, Local Search, Optimization, Resource selection

ACM Reference Format:
Federica Filippini, Hamta Sedghani, and Danilo Ardagna. 2023. SPACE4AI-R: a Runtime Management Tool for AI Applications Component Placement and Resource Scaling in Computing Continua. In *2023 IEEE/ACM 16th International Conference on Utility and Cloud Computing (UCC '23), December 4–7, 2023, Taormina (Messina), Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3603166.3632560

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs International 4.0 License.

UCC '23, December 4–7, 2023, Taormina (Messina), Italy

© 2023 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0234-1/23/12.
https://doi.org/10.1145/3603166.3632560

1 Introduction

The Computing Continuum paradigm has recently emerged as the natural intersection between Edge Computing, characterized by resource-constrained devices guaranteeing low latencies thanks to the proximity to data-generating sensors [24], and Cloud Computing, which makes accessible an ideally-unlimited computational power at the price of high network communications overheads [18]. According to this model, Artificial Intelligence (AI) applications can be executed by deploying latency-sensitive tasks on Edge devices, while resource-intensive tasks are offloaded to the Cloud [5]. This generates a fragmented scenario where the computing and storage capabilities are distributed among devices with highly heterogeneous capacities. Effective methods are needed to orchestrate at best the Computing Continuum resources, and to determine the optimal placement for AI applications components minimizing the expected costs while meeting Quality of Service requirements (see Figure 1).

This Resource Selection and Component Placement (RS-CP) problem should be tackled in two different phases: at design time and at runtime. Design-time algorithms aim to optimize the initial placement, before the application execution starts, based on the expected input workload and the predicted components performance. This may become sub-optimal over time, since the actual workload is usually subject to fluctuations due, for instance, to variations in the generated data volumes. Thus, the optimal solution has to be monitored and adapted dynamically while the application is running. While a design-time tool is allowed to take as much time as needed (up to several minutes) to find the initial placement, a runtime tool, which is executed online, must provide a feasible reconfiguration in few seconds at most.

Building upon a state-of-the-art design-time solution [19], we propose SPACE4AI-R: a tool to effectively address the RS-CP problem at runtime, supporting the execution of AI applications on Computing Continuum resources including Edge devices, Cloud Virtual Machines and Function as

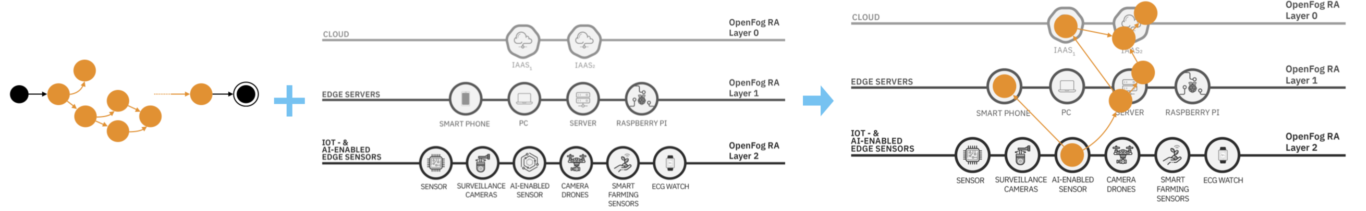


Figure 1. The placement problem: mapping an application on the infrastructure

a Service configurations. Through a Random Search combined with a Stochastic Local Search heuristic, SPACE4AI-R copes with dynamic workload fluctuations, identifying the minimum-cost reconfiguration of the running placement. It is worth noting that, unlike standard software components, AI-based components consisting of DNN are partitionable. SPACE4AI-R effectively tackle this scenario by identifying also the most suitable NN deployment for these components.

Experimental results show that our tool can yield up to 60% cost reductions with respect to a static placement in a realistic use-case concerning the identification of wind-turbine blades damage, and it tackles large-scale systems with up to 15 application components in less than 1.5s.

The rest of the paper is organized as follows: Section 2 describes the RS-CP problem and the optimization model developed to characterize it. Section 3 describes the SPACE4AI-R framework, whose experimental validation is reported in Section 4. Section 5 briefly overviews the state of the art. Conclusions are finally drawn in Section 6.

2 RS-CP Problem

This section describes the application model (Section 2.1), the computing infrastructure (Section 2.2), and the mathematical formulation developed to characterize the RS-CP problem (Section 2.3).

2.1 Application Model

In our framework, AI applications are modeled as Directed Acyclic Graphs (DAGs), as in Figure 2, whose nodes represent the different application components. We assume that each component is a Deep Neural Network (DNN) function that can be run in a Docker container deployed either in an Edge device, in a Cloud virtual machine (VM) or using the Function as a Service (FaaS) paradigm.

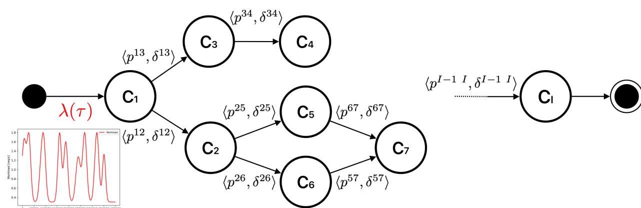


Figure 2. Application DAG

We assume that the DAG includes a single entry point, characterized by an exogenous input workload $\lambda(\tau)$ (expressed in terms of requests/sec) that varies over time, and a single exit point. The set of all application components is denoted by $\mathcal{I}$. The directed edge from a node i to a node k is labelled with $\langle p^{ik}, \delta^{ik} \rangle$, where p^{ik} is the transition probability between component i and component k, while δ^{ik} is the transferred data size. Each $i \in \mathcal{I}$ is characterized by a load $\lambda^i(\tau)$ that depends on the exogenous workload $\lambda(\tau)$ and on the transition probabilities from the previous components.

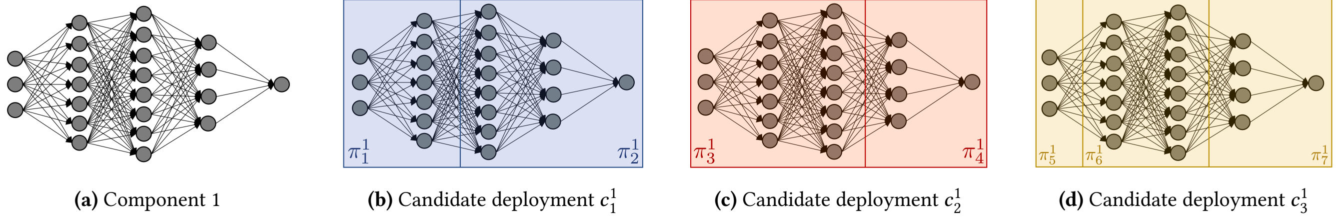
As already mentioned, DNNs may be partitioned differently according to resources capacity and network settings [15]. Thus, we say that components are characterized by multiple candidate deployments, denoted by the set C^i. Note that these partitions can be filtered, depending on the tensors size and a detailed profiling, to select only the candidate deployments that reduce the transmission latency and energy for transmission. However, this is not explicitly mentioned in our work because it is usually performed at design-time (see, e.g., [13]).

Each element $c_s^i \in C^i$ represents a different way of partitioning the DNN, i.e., it is defined as $c_s^i = \{\pi_h^i\}_{h \in \mathcal{H}_s^i}$, where π_h^i denotes the single partition and $\mathcal{H}_s^i$ is the set of all partition indices (see Figure 3).

The main performance metric we consider in our framework is the response time. Quality of Service (QoS) requirements might be imposed on both the response time of single components (*local constraints*), and on the response time of a sequence of consecutive components, denoted as path (*global constraints*).

2.2 Infrastructure

Computing Continuum resources include Edge devices, Cloud Virtual Machines (VMs) and Function as a Service (FaaS) configurations. In particular, FaaS is a type of Cloud computing service that provides a serverless paradigm to execute tasks, implemented as functions, on a Cloud platform. We denote the set of the candidate Edge resources by $\mathcal{J}_\mathcal{E} = \{1, ..., E\}$, the set of the candidate Cloud devices by $\mathcal{J}_\mathcal{C} = \{E + 1, ..., E + C\}$ and the set of all FaaS configurations by $\mathcal{J}_\mathcal{F} = \{E + C + 1, ..., E + C + F\}$, so that $\mathcal{J} = \mathcal{J}_\mathcal{E} \cup \mathcal{J}_\mathcal{C} \cup \mathcal{J}_\mathcal{F}$. Furthermore, we denote by $n_j \in \mathbb{N}$ the maximum number of available instances for all resources $j \in \mathcal{J}_\mathcal{E} \cup \mathcal{J}_\mathcal{C}$. We introduce the concept of *computational layers*, defined as disjoint sets of resources that can be considered as alternative to one another



(a) Component 1 **(b)** Candidate deployment c_1^1 **(c)** Candidate deployment c_2^1 **(d)** Candidate deployment c_3^1

Figure 3. Example of AI application component with its candidate deployments

(e.g., different VM types from the same Cloud provider's catalog, or two alternative models for a specific Edge device). Only one resource can be selected in each layer.

To identify the optimal placement, we need to determine which resource types $j \in \mathcal{J}$ are compatible with each partition π_h^i for all the candidate deployments c_s^i characterizing component $i \in \mathcal{I}$ (this depends on the hardware characteristics and available network connections). Hence, we introduce a compatibility matrix $\mathbf{A} = [a_{hj}^i]$, where a_{hj}^i is 1 if π_h^i can be executed on resource j. Moreover, each partition π_h^i is characterized by a different memory requirement $\tilde{m}_{hj}^i$ depending on the resource where it is allocated, and each resource is characterized by a memory limit M_j.

To compute the response time of the component partitions involved in *local* and/or *global* constraints, we exploit different models. For Edge resources and Cloud VMs we adopt either the queuing theory, representing each resource as single server multiple class queue system (i.e., as an individual M/G/1 queue), or Machine Learning (ML)-based models developed as in [9]. For FaaS configurations, we rely either on similar ML-based models or on an external tool proposed in [16], which allows to compute the response time of a function depending on various parameters, such as the requests arrival rate, the demanding time, and the expiration time before the function shutdown. Additionally, we consider the network delays induced by data transmissions between partitions and components executed on different resources. The communication between the different kinds of devices is enabled by several network domains, exploiting different technologies (e.g., WiFi, 5G), each one characterized by the corresponding access time and bandwidth.

Finally, we assume that Edge resources are characterized by energy-consumption hourly costs, Cloud VMs follow a per-second billing that depends on the chosen provider, while FaaS costs are expressed in GigaByte-second, and they depend on the memory size, the functions duration, the total number of invocations. We indicate with $Cost_E$, $Cost_C$, and $Cost_F$ the total costs of Edge devices, Cloud VMs and FaaS configurations, respectively.

2.3 Problem Statement

The RS-CP problem can be formulated as a Mixed Integer Non-Linear Program (MINLP) aiming at minimizing the placement cost while satisfying hardware, network and QoS constraints. While this formulation is built upon the one presented in [19], physical and virtual computational layers are managed differently at design time and runtime. Indeed, the resource type in a physical layer is selected in the initial, design-time solution and it cannot be modified in future reconfigurations, albeit scaling actions are allowed. Virtual resources, instead, can be scaled to zero and eventually replaced by other types if required, since their provisioning time is usually not too large. Therefore, determining a solution for the MINLP problem at runtime means: (i) switching on/off or scaling the the appropriate Edge devices, which are physical resources already selected at design-time; (ii) scaling in/out the number of VM instances, or selecting the optimal type for virtual resources that are currently unused; (iii) identifying a deployment for each component; (iv) allocating the partitions on the chosen devices; (v) checking if the assignments are compatible with memory constraints and QoS requirements. To characterize which resources should be selected and how partitions should be assigned to the available devices, we introduce the following variables: x_j, which is 1 if device $j \in \mathcal{J}$ is used in the final deployment; y_{hj}^i, which is equal to 1 if partition π_h^i of candidate deployment c_s^i is deployed on resource j; $\widehat{y}_{hj}^i$, which denotes the number of resource instances of type j assigned to any partition π_h^i. These allow to quantify the operational costs of the resources and to define the objective function of our problem, namely:

$$\min Cost_E + Cost_C + Cost_F,$$

subject to assignment compatibility, QoS requirements, memory and allocation constraints.

3 SPACE4AI-R

Finding the optimal placement of an application over an infrastructure is very challenging. Indeed, mathematical formulations proved to be NP-hard, and heuristic algorithms are usually exploited to solve this problem. In this work, we propose a Random Search (RS) combined with Stochastic Local Search (SLS) algorithm.

RS (see Algorithm 1) aims at determining a pool K of good-quality solutions (i.e., feasible deployments with low execution costs) by randomly exploring the search space. In particular, new candidates are iteratively constructed by selecting at random the resources to be considered (line 5),

the components deployments (line 7), and the partition-to-resource assignments (line 9). To increase the probability of generating a feasible solution, the number of resource instances is initially set to the maximum (line 13), and then tentatively reduced to lower the costs (lines 15–17).

Algorithm 1 Random Search

```
1:  Input: I,H,J,DAG, A, Demanding time, QoS constraints, Resources
    costs, MaxIter, K
2:  Initialization: Solutions ← ∅
3:  for 1, ..., MaxIter do
4:      x ← [0], y ← [0], ŷ ← [0]
5:      Randomly pick a node j at each layer; set x_j ← 1
6:      for i ∈ I do
7:          Randomly pick a deployment c_s^i ∈ C
8:          for h ∈ H_s^i do
9:              Randomly assign partition π_h^i to a node j s.t. x_j = 1 and
                a_hj^i = 1 and set y_hj^i ← 1, ŷ_hj^i ← 1
10:         end for
11:     end for
12:     for ∀j ∈ J_E ∪ J_C such that y_hj^i = 1 do
13:         ŷ_hj^i ← n_j ∀i ∈ I, ∀h ∈ H_s^i
14:     end for
15:     if solution ⟨x, y, ŷ⟩ is feasible then
16:         for ∀j ∈ J_E ∪ J_C such that ŷ_hj^i > 1 do
17:             ReduceClusterSize(j)
18:         end for
19:         Solutions ← Solutions ∪ ⟨x, y, ŷ⟩
20:     end if
21: end for
22: if Solutions ≠ ∅ then
23:     Compute cost for all solutions
24:     Sort solutions by cost
25:     return k best Solutions {the top K solutions with lower costs}
26: else
27:     return Solutions {No feasible solution found}
28: end if
```

SLS (see Algorithm 2) starts from the K candidates found by RS and stochastically explores their *neighborhoods*, which are reached through a predefined set of *moves*, to improve the solution quality. The set of moves $\mathcal{T}$ includes atomic changes to the initial solutions defined by, e.g, migrating a partition running on FaaS to an already-deployed VM, changing a FaaS configuration with another one of lower cost (i.e., characterized by less memory), or changing the deployment chosen for a component. Moreover, $\mathcal{T}$ includes more complex moves, e.g., dropping an Edge server or Cloud VM from the placement by allocating its partitions on the remaining running resources, or changing the Edge server or the VM with another node with smaller cost.

4 Experimental Results

The experimental analysis is divided into two parts. The first one (Section 4.1) deals with a real use-case application related to the inspection of wind farms. The second part (Section 4.2) deals with the scalability analysis, where we prove that our tool is able to tackle small to large-scale general workflows. For both analyses, we simulated a dynamic workload by considering a bi-modal profile for the duration of the application (fixed to be 2 hours in our simulations), and assume a periodic system reconfiguration (every 5 minutes). We set $\lambda \in [\lambda_{min}, \lambda_{max}]$ req/s and compute the design-time solution for $\lambda(0) = \lambda_{max}$. This is a reasonable assumption, since sizing the system for the worst case is the most conservative option to avoid violating QoS requirements. Moreover, this choice guarantees that a feasible solution exists for the entire application execution; indeed, Edge resources, which have usually less computational capacity, may become the bottleneck if the load increases more than expected. The considered workload profiles are reported in Figure 5.

Algorithm 2 Stochastic Local Search

```
1:  Input: DAG, H, J, A, QoS constraints, costs, MaxIter, RS_sols
2:  Initialization: BestSol ← Best among RS_sols
3:  for s ∈ RS_sols do
4:      CurrSol ← s
5:      NewSol ← s
6:      for n = 1, ..., MaxIter do
7:          Randomly pick T ∈ T
8:          NewSol ← T(CurrSol)
9:          if Cost(NewSol) < Cost(CurrSol) and NewSol is feasible
            then
10:             CurrSol ← NewSol
11:         end if
12:     end for
13:     if Cost(CurrSol) < Cost(BestSol) then
14:         BestSol ← CurrSol
15:     end if
16: end for
17: return BestSol
```

The algorithms 1 and 2 were implemented in C++. The experiments were executed on a HP Probook 455 with 1.9/4.4 GHz CPU AMD Ryzen 7 5800U and 16GB of memory.

The source code, configuration files and results of all the experiments presented here are available at [8].

4.1 Use-case analysis

The DAG of the use-case application we considered in this section is reported in Figure 4. It includes 7 components, which progressively process images collected by drones to identify and classify damages in wind turbines blades. The components can be executed on different Computing Continuum resources, grouped in four computational layers (the component-to-resource compatibility is represented by dotted arrows), and the initial placement selected at design-time, called *production deployment*, is indicated by red arrows.

We design three testing scenarios, which differ in the value of λ_{max} (see Figure 5a) and in the type of resource selected at design time in computational layer 2 (see Figure 4): **(A)** user's PC, $\lambda_{max} = 1.8$ req/s; **(B)** 2 servers in the user's van, $\lambda_{max} = 7.5$ req/s; **(C)** 3 Mobile Edge Computing (MEC) servers accessed from a nearby 5G tower, $\lambda_{max} = 18$ req/s.

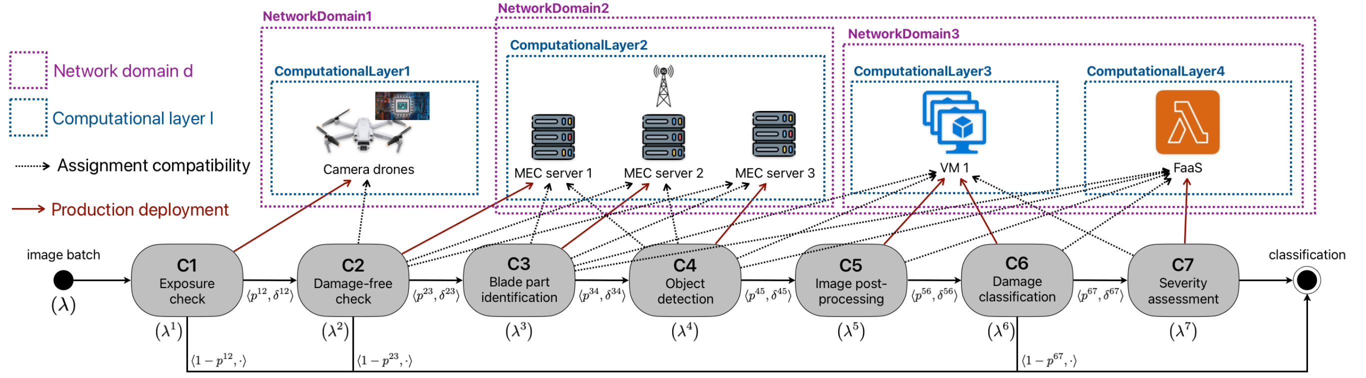
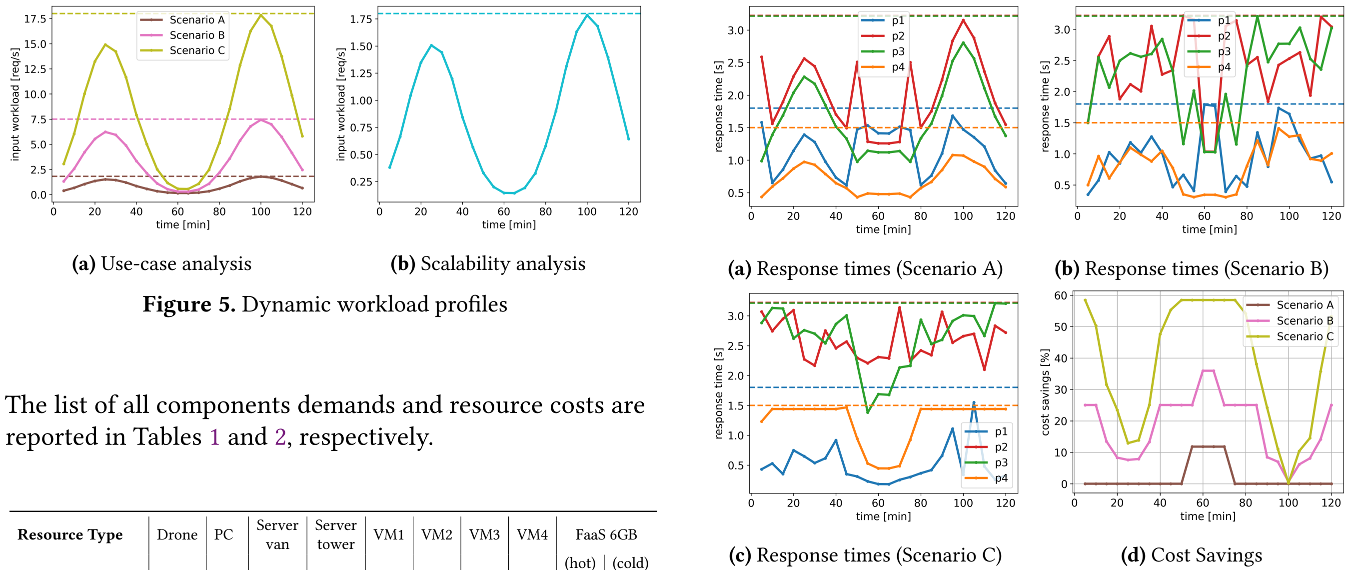


Figure 4. A use case of identifying wind turbines blade damage (Scenario C)



(a) Use-case analysis **(b)** Scalability analysis

Figure 5. Dynamic workload profiles

(a) Response times (Scenario A) **(b)** Response times (Scenario B)

(c) Response times (Scenario C) **(d)** Cost Savings

Figure 6. Path response times and dynamic placement cost savings

The list of all components demands and resource costs are reported in Tables 1 and 2, respectively.

Resource Type			Drone	PC	Server van	Server tower	VM1	VM2	VM3	VM4	FaaS 6GB (hot)	FaaS 6GB (cold)
Demand [s]	C_1	h_1	0.01	-	-	-	-	-	-	-	-	-
	C_2	h_1	1.76	0.52	0.25	0.13	-	-	-	-	-	-
		h_2	0.52	0.20	0.10	0.05	-	-	-	-	-	-
		h_3	0.52	0.23	0.12	0.06	-	-	-	-	-	-
		h_4	0.69	0.25	0.13	0.07	-	-	-	-	-	-
	C_3	h_1	-	1.52	0.95	0.73	1.43	0.46	0.34	0.19	1.27	2.21
		h_2	-	0.38	0.24	0.19	0.37	0.10	0.11	0.06	1.25	2.10
		h_3	-	0.38	0.24	0.18	0.36	0.10	0.11	0.05	1.05	1.94
		h_4	-	0.39	0.24	0.18	0.36	0.10	0.10	0.05	1.19	1.85
		h_5	-	0.38	0.24	0.18	0.36	0.10	0.10	0.05	1.14	2.19
		h_6	-	0.50	0.32	0.24	0.48	0.13	0.12	0.07	1.41	2.27
		h_7	-	0.50	0.32	0.24	0.48	0.13	0.12	0.07	1.14	1.62
		h_8	-	0.49	0.32	0.24	0.48	0.13	0.12	0.07	1.38	2.16
	C_4	h_1	-	1.36	0.88	0.65	1.32	0.33	0.33	0.18	1.01	1.54
	C_5	h_1	-	-	-	-	1.63	0.40	0.27	0.20	1.58	2.27
		h_2	-	-	-	-	0.41	0.11	0.07	0.05	1.25	1.90
		h_3	-	-	-	-	0.41	0.11	0.07	0.05	1.37	1.97
		h_4	-	-	-	-	0.42	0.11	0.07	0.05	1.20	2.10
		h_5	-	-	-	-	0.42	0.11	0.07	0.05	1.35	1.89
		h_6	-	-	-	-	0.55	0.14	0.10	0.08	1.32	1.83
		h_7	-	-	-	-	0.55	0.14	0.10	0.08	1.05	1.79
		h_8	-	-	-	-	0.55	0.14	0.10	0.08	1.46	2.30
	C_6	h_1	-	-	-	-	1.59	0.33	0.33	0.20	1.04	1.70
	C_7	h_1	-	-	-	-	1.47	0.45	0.31	0.22	1.44	2.23
		h_2	-	-	-	-	0.74	0.23	0.16	0.12	1.59	2.40
		h_3	-	-	-	-	0.74	0.23	0.16	0.12	1.51	2.11

Table 1. Use-case analysis: demand of component partitions on the compatible resources

The outcomes achieved running both RS and LS for 10^4 iterations are reported in Figure 6. In each scenario, we imposed four QoS constraints on different components paths: in particular, we prescribed that the total response time of the first two components ($p1$) must not exceed $1.8s$, the total response time of $p2 = \{C_3, C_4, C_5\}$ and $p3 = \{C_4, C_5, C_6\}$ must not exceed $3.2s$, and the total response time of the last component ($p4 \equiv C_7$) must not exceed $1.5s$. Figures 6a, 6b and 6c report the values observed during the whole simulation, for the three scenarios, together with the constraints thresholds (dashed lines). We can observe that, in Scenario A,

Name	Cost [$/h]	Number of Instances
VM1	0.41	n=4
VM2	1.53	n=3
VM3	1.99	n=3
VM4	3.16	n=3

Table 2. Use-case analysis: Cloud resources

the response times are always quite far from the threshold, except, for path $p1$, in a central area where, as reported in Figure 5a, the workload approaches λ_{min}. Indeed, when this happens SPACE4AI-R suggests to switch off the PC in layer 2 and execute both C_1 and C_2 on the drone, increasing its utilization. A similar pattern occurs in Scenario B, where, however, it is more difficult for SPACE4AI-R to determine feasible solutions due to the higher workload. In particular, we can note that the response times are generally closer to the thresholds. Finally, the response times are more stable in Scenario C, where only $p4$ is always close to the threshold. Figure 6d reports the cost savings yielded by dynamically reconfiguring the system instead of considering a static allocation, where the initial design-time solution determined as in [19] is kept fixed throughout the application execution. The runtime management provides a cost reduction in all the considered scenarios, up to 60% when the computing infrastructure is more complex. The savings are more significant when the input workload is smaller, since the design-time solutions were determined in a maximum-load condition.

The average time required by SPACE4AI-R to compute each reconfiguration in the three scenarios is of $0.396s$, $0.412s$ and $0.427s$, respectively (std. dev.: $0.081s$, $0.084s$ and $0.088s$).

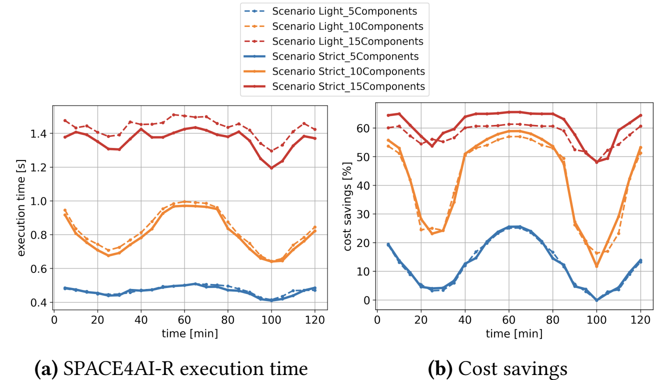
4.2 Scalability analysis

To evaluate the scalability of our approach, we consider three different scenarios at different scale, namely with 5, 10 and 15 components including up to 3 candidate deployments each and up to 4 partitions per deployment. It is worth noting that, since the placement occurs at the level of the component partitions, tackling the largest scenario means allocating on average 75 objects (considering 2 deployments per component, and 2.5 partitions). Service demands where randomly generated in the range of $[1, 5]s$ for Edge resources, $[0.5, 2]s$ for VMs, and $[2, 5]s$ for cold and warm FaaS requests, according to other literature proposals [6, 17, 19]. For each scenario, we randomly generated 10 DAGs with branches, exploiting Networkx [11], to check how our tool deals with general workflows. Furthermore, we consider two ranges of values for the local and global constraints, namely *strict* (maximum response times are set very close to the resources demands) and *light* (components and paths response time thresholds are set very high). These distinction allows to assess how the tool deals with opposite conditions, where determining a feasible reconfiguration is more or less challenging. The number and type of resources and the number of local and global constraints in each scenario is reported in Table 3. Local and global constraints thresholds were randomly chosen in the ranges of $[50, 100]s$ and $[200, 300]s$, respectively, for the light and strict-constraints scenario. Similarly, they were set in the ranges of $[7, 10]s$ and $[20, 25]s$ in the case of strict constraints.

Experiments show that SPACE4AI-R succeeds in all the scenarios, for each time instant and DAG. Figure 7 shows that, averaging the results across the 10 random DAGs, the tool tackles even large instances in the order of seconds, which is close to 100× faster than the design-time tool, while guaranteeing cost reductions against the static placement, meeting the reactivity requirements of a runtime management framework. Note that the cost savings are higher for larger systems, that benefit more from the reconfigurations. The SPACE4AI-R execution time is larger when the workload is low and in the light-constraints scenario, since more candidate feasible solutions can be generated and explored in these settings.

Scenario		1	2	3
Number of components		5	10	15
Type and number of resources in each layer	CL_1	Drone: 1	Drone: 1	Drone: 1
	CL_2	Edge: 2	Edge: 4	Edge: 5
	CL_3	VM: 3	Edge: 4	Edge: 5
	CL_4	FaaS: 2	VM: 4	VM: 5
	CL_5	-	VM: 4	VM: 5
	CL_6	-	FaaS: 4	VM: 5
	CL_7	-	-	FaaS: 5
Number of local, global constraints		3,3	4,4	5,5

Table 3. Scalability analysis scenarios



(a) SPACE4AI-R execution time **(b)** Cost savings

Figure 7. Scalability analysis

5 Related Work

Placement problems are continuously gaining attention from the research community. A classification of the literature proposals in terms of the placement purpose (e.g., scheduling, offloading, distribution of physical resources), the computing paradigm (Cloud-Edge, only Cloud, only Edge), and the optimization metrics (latency time, energy consumption, cost) is proposed in [2]. Similarly, [22] reviews the placement methods according to the infrastructure and applications characteristics. Most of the proposals [23, 7, 12, 1, 3, 21] focus on a design-time perspective, tackling the resource allocation and optimal component partitioning under a fixed workload ([23]), minimizing the total processing time by task offloading in an Edge-to-Cloud infrastructure ([7]), or dealing with energy optimization ([3, 21]). Among the design-time approaches, [12, 1] are the closest to our work: in [12], the computing infrastructure is represented by an Undirected

Graph, where nodes correspond to Edge and Cloud clusters, while the application is modeled as a DAG. They aim to minimize the total processing, memory and data-transfer costs through a greedy approach, splitting the application into subsets of *star* components sorted by the number of connection links and then selecting the best provisioning for each star component. In [1], the multi-component application placement problem is tackled via two heuristic algorithms: a matching and local search-based method that is very efficient when the number of components and devices is relatively limited, and a greedy algorithm designed for larger systems.

However, runtime tools are becoming crucial, since demands are subject to fluctuations and finding statistical knowledge of future requests is quite challenging. [4] presents an ML-based auto-scaling system that can behave proactively or reactively to adjust the number of Edge nodes in response to workload changes. [10] addresses the service offloading and placement in the Computing Continuum through a greedy algorithm based on the online demands prediction. [20] proposes a general online orchestration tool that deals with dynamic workloads in different computing environments without any prior assumption on the future system states and future demand trends. [14] proposes an online knapsack method for the dynamic placement and migration of AI workflows under latency constraints.

To the best of our knowledge, the runtime tools presented in existing literature have predominantly focused on a single application instance, neglecting any consideration of resource contention in the the assessment of application performance. The uniqueness of our paper lies in our commitment to factoring in resource contention when estimating application performance.

6 Conclusions

This work proposes SPACE4AI-R, a tool to support AI application component placement and resource selection at runtime. Thanks to the fast heuristics and efficient implementation, our approach is able to cope with dynamic workloads from small to large scale systems, exhibiting execution times under 1.5s and two order of magnitudes faster than the design-time approach, guaranteeing up to 60% cost savings over a static placement. Future works will extend the Stochastic Local Search algorithm by investigating other neighborhood exploration techniques.

Acknowledgment

This work has been funded by the European Commission under the H2020 grant N. 101016577 AI-SPRINT: AI in Secure Privacy pReserving computINg conTinuum.

References

[1] Tayebeh Bahreini and Daniel Grosu. 2022. Efficient Algorithms for Multi-Component Application Placement in Mobile Edge Computing. *IEEE Trans. Cloud Comput.*, 10, 4, 2550–2563.

[2] Julian Bellendorf and Zoltán Ádám Mann. 2020. Classification of optimization problems in fog computing. *Future Gener. Comput. Syst.*, 107, 158–176.

[3] Jing Bi et al. 2023. Energy-Efficient Computation Offloading for Static and Dynamic Applications in Hybrid Mobile Edge Cloud System. *IEEE Trans. Sustain. Comput.*, 8, 02, 232–244.

[4] Thiago Pereira da Silva et al. 2022. Online machine learning for auto-scaling in the edge computing. *Pervasive Mob.*, 87, 101722.

[5] Sijing Duan et al. 2023. Distributed artificial intelligence empowered by end-edge-cloud computing: a survey. *Commun. Surveys Tuts.*, 25, 1, 591–624.

[6] Tarek Elgamal et al. 2018. Costless: optimizing cost of serverless computing through function fusion and placement. In *IEEE/ACM SEC*, 300–312.

[7] Wenhao Fan et al. 2022. Collaborative Service Placement, Task Scheduling, and Resource Allocation for Task Offloading with Edge-Cloud Cooperation. *IEEE Trans. Mob. Comput.*, 1–18.

[8] Federica Filippini et al. AI-SPRINT SPACE4AI-R Local Search. Zenodo, (Dec. 2022). DOI: 10.5281/zenodo.7437888.

[9] E. Galimberti et al. [n. d.] OSCAR-P and amllibrary: performance profiling and prediction of computing continua applications. In *ACM/SPEC ICPE 2023*, 139–146.

[10] Yeting Guo et al. 2022. PARA: Performability-aware resource allocation on the edges for cloud-native services. *Int. J. Intell. Syst.*, 37, 11, 8523–8547.

[11] Aric A. Hagberg et al. 2008. Exploring network structure, dynamics, and function using networkx. In *SciPy Proceedings.* Gaël Varoquaux et al., (Eds.) Pasadena, CA USA, 11–15.

[12] Baudouin Herlicq et al. 2022. NextGenEMO: an Efficient Provisioning of Edge-Native Applications. In *IEEE ICC 2022*, 1924–1929.

[13] Yiping Kang et al. 2017. Neurosurgeon: collaborative intelligence between the cloud and mobile edge. *SIGARCH Comput. Archit. News*, 45, 1, 615–629.

[14] Qianlin Liang et al. 2023. Model-driven cluster resource management for ai workloads in edge clouds. *ACM Trans. Auton. Adapt. Syst.*, 18, 1, Article 2.

[15] Guozhi Liu et al. 2023. An adaptive dnn inference acceleration framework with end–edge–cloud collaborative computing. *Future Gener. Comput. Syst.*, 140, 422–435.

[16] Nima Mahmoudi and Hamzeh Khazaei. 2022. Performance Modeling of Serverless Computing Platforms. *IEEE Trans. Cloud Comput.*, 10, 4, 2834–2847.

[17] Johannes Manner et al. 2018. Cold start influencing factors in function as a service. In *IEEE/ACM UCC Companion*, 181–188.

[18] P. Mell and G. Timothy. 2011. SP 800-145. The NIST Definition of Cloud Computing. Tech. rep. Gaithersburg, MD, USA.

[19] Hamta Sedghani et al. 2021. A random greedy based design time tool for ai applications component placement and resource selection in computing continua. In *IEEE EDGE*, 32–40.

[20] Xun Shao et al. 2023. An Online Orchestration Mechanism for General-Purpose Edge Computing. *IEEE Trans. Serv. Comput.*, 16, 02, 927–940.

[21] Ying Chen Shaoxuan Yun. 2023. Intelligent Traffic Scheduling for Mobile Edge Computing in IoT via Deep Learning. *CMES*, 134, 3, 1815–1835.

[22] Sven Smolka and Zolt Mann. 2022. Evaluation of Fog Application Placement Algorithms: A Survey. *Computing*, 104, 6.

[23] Yi Su et al. 2023. Joint DNN Partition and Resource Allocation Optimization for Energy-Constrained Hierarchical Edge-Cloud Systems. *IEEE Trans. Veh. Technol.*, 72, 3, 3930–3944.

[24] Sheng Yue et al. 2022. Todg: distributed task offloading with delay guarantees for edge computing. *IEEE TPDS*, 33, 7, 1650–1665.