Security-Aware Resource Allocation in the Edge-Cloud Continuum

Polyzois Soumplis[1,2], Georgios Kontos[1,2], Aristotelis Kretsis[1,2], Panagiotis Kokkinos[2,3], Anastassios Nanos[4] and Emmanouel Varvarigos[1,2]
[1] School of Electrical & Computer Engineering. National Technical University of Athens, Greece
[2] Institute of Communication and Computer Systems, Athens, Greece
[3] Department of Digital Systems, University of Peloponnese, Sparta, Greece
[4] Nubificus LTD, Sheffield, United Kingdom

***Abstract*—Cloud-native applications, comprised of multiple services, optimize their execution on the edge cloud continuum, by leveraging both edge for time-critical computations and the more distant (but abundant) cloud resources for not time-driven computations. The infrastructure is controlled by a hierarchical orchestrator logic, with sub-modules at each level managing a specific set of resources. A crucial challenge in deploying applications over such a distributed infrastructure is the allocation of resources, considering jointly application-specific security requirements and computing and networking constraints, that increase significantly the complexity of the decision-making process. In this work, we assume varying levels of workload isolation achievable through lightweight virtualization mechanisms, establishing distinct tiers of security and trustworthiness, each with its own quantified computational and storage requirements. We model the respective resource allocation problem, i.e., of provisioning edge-cloud continuum resources for cloud-native applications subject to applications' performance and security requirements, as a Mixed Integer Linear Program. Additionally, a best-fit heuristic is introduced to reduce the execution time for real-size scenarios, performing a fast allocation of resources for the applications while maintaining a tolerable optimality gap. Finally, a Multi-agent Rollout Mechanism is proposed that trades off execution time with performance leveraging the greedy heuristic for the approximation of future decisions in a Reinforcement Learning manner. Several simulation experiments were performed to showcase the effectiveness of the developed mechanisms, while simultaneously addressing the needs of conflicting objectives.**

Keywords— resource allocation, cloud-native applications, cloud edge continuum, security, workload isolation

I. INTRODUCTION

We are witnessing a wave of emerging cloud computing technologies and services that empower advanced applications from different vertical sectors with diverse requirements. Centralized cloud computing infrastructures and respective services currently handle most of these applications' processing and storage requirements. However, it has been recognized that only a fragment of the data generated will be truly useful, while their size will exceed the storage capabilities of today's cloud datacenters [1], [2]. At the same time, the application's performance requirements become increasingly strict, e.g., in terms of latency, geographic density, security and energy consumption.

To address these requirements, computation and storage resources are placed at the network periphery in a concept known as edge computing. The realization of an edge-cloud continuum has been recognized as key to overcoming the bottlenecks in data collection and transmission present in centralized processing, as it can improve manifold the applications' execution latency, the use of networking, and the efficiency in the infrastructures' utilization [3], [4], reducing also networking congestion.

Cloud-native applications, characterized by their modularity and scalability, operate in dynamic, multi-technology environments. They consist of loosely coupled small, independent components known as microservices. These components offer increased flexibility and auto-scalability, but present unique computing and security requirements. Their independent deployment across the edge-cloud continuum adds complexity to the application rollout process. Deployment challenges include managing microservice communication and service chain embedding, necessitating secure execution amongst concurrently running microservices on identical resources.

As a potential solution to the demanding requirements of deploying microservices in a cloud-edge environment, the concept of virtualization has emerged. Virtualization can facilitate isolated execution, but it can also increase the overhead and reduce the already limited computing and storage capacity of edge devices. Instead of using a full virtualization stack with a hypervisor and virtual machines, OS-level virtualization through containers can be a better option [5]. It significantly reduces the overhead associated with the traditional Virtual Machine stack: VMM (e.g., QEMU), hardware extensions (KVM). However, this approach has a drawback, containers share the same OS kernel and a malicious application running in a container can compromise the entire system. A solution is to provide edge resources with security and trustworthiness tiers.

Allocating resources in a distributed multi-tenant infrastructure poses challenges for a centralized orchestrator. In order to address these challenges, hierarchical orchestrator architectures are employed to enable a more efficient resource allocation. SERRANO [6] adopts a similar approach, with high-level decision taken, by the Resource Orchestrator, on the edge-cloud continuum layer, and low-level scheduling performed by each platform's orchestration mechanisms (i.e., Local Orchestrator). This provides several degrees of freedom to the Local Orchestrator for serving in an optimal manner the "request", satisfying both the central orchestrator and the resource's requirements.

In this work, our primary focus is on the efficient allocation of resources for cloud-native applications within multitenant edge and cloud infrastructures. Our approach encompasses both the intricacies involved in orchestrating inter-communicating microservices and the necessity of fulfilling their security execution requirements. We evaluate different degrees of workload isolation and trusted execution, exploring cutting-edge technologies such as sandboxing and unikernels and we quantify the resulting trade-offs in the computing and storage requirements. Then we develop three

different security-aware resource allocation mechanisms, initially modelling the respective resource allocation problem as a Mixed Integer Linear Program (MILP). Recognizing the vast search space for real-sized application deployment and its long execution time, we propose a greedy heuristic and a multi-agent rollout approach. The latter mechanism is based on the principles of the approximate dynamic programming and reinforcement learning in order to exploit efficiently the trade-off between execution time and resource allocation efficiency. The rest of this paper is organized as follows. In section II, we report on related work. Section III presents the considered infrastructure. Section IV describes the resource allocation mechanisms. Section V evaluates the efficiency of these mechanisms and finally Section VI concludes our work.

II. Related Work

Numerous studies have explored the resource allocation problem for the placement of applications over computing and networking infrastructures. The authors in [7] examine the placement of virtual machines on top of physical systems in a cloud data center to perform big-data analytics from IoT devices, targeting to optimize the utilization of network resources. In [8], the authors develop an algorithm based on Gaussian Process Regression in order to predict future traffic and minimize request blocking, especially in the case of time critical requests. They consider a hierarchical infrastructure, and the algorithm is used to ensure the lower layer resource sufficiency for future time-sensitive demands. The authors in [9] present Foggy, an architectural framework based on open-source tools that handles requests from end users in a multi-level heterogenous Fog environment and serves them in a greedy, best-fit approach, based on their computing needs.

To enable the secure execution of cloud native applications, frameworks are introduced [10] that support container execution in sandboxed environment based on micro-VMs. Recent works also recommend unikernels [11], [12], which are specialized machine images, tailored to a single application. Unikernels have minimal memory/system footprint, achieve high performance and provide strong isolation equivalent to that of virtual machines. These trends give rise to a number of fundamental challenges that relate to the application deployment, the support of heterogeneous infrastructures, and the provided security. The authors in [13] focus on the challenges and requirements for building a scalable and trustworthy multi-tenant AIoT (Artificial Intelligence of Things) cloud-native platform. They first identify several key challenges, including security, privacy and trust and highlight how these challenges differ in a multi-tenant edge environment in comparison with a central cloud. They also present the state-of-the-art methods for addressing these challenges and describe open research areas.

In [14], the authors propose a security-aware dynamic scheduling approach for cloud-based industrial applications in a two-tier infrastructure. They introduce a three-level security model that corresponds to public, semi-public and private data. A distributed Particle Swarm Optimization heuristic is developed to perform resource allocation, and a dynamic scheduling mechanism to deal with real-time optimization. The authors in [15] propose a security-aware offloading model for a multi-user environment. A new security layer is introduced utilizing the Advanced Encryption Standard cryptographic algorithm to prevent attacks such as sniffing, jamming and eavesdropping. The resource allocation problem is formulated with the optimization objective of minimizing the latency and energy overhead of mobile users, leveraging a Deep Reinforcement learning algorithm.

The work in [16] presents a security-aware task offloading method for maximizing the total profit of edge nodes in an Edge-Cloud computing (ECC) environment. A security model is constructed, utilizing several confidentialities (e.g., AES) and integrity (e.g., SHA1, MD5) services for coping with security threats. A genetic algorithm is developed to solve the respective resource-allocation problem. The authors in [17] discuss the evolution of machine learning from a centralized approach to distributed machine learning (DML) and federated learning (FL) in terms of data privacy and security. They argue that FL is more secure and privacy-preserving than DML, as FL allows data to remain on local devices rather than being centrally collected and processed.

In our work, to enable the efficient execution of cloud native applications over the edge-cloud continuum, we model the dependencies among the microservices along with their communication requirements. In addition, we consider application isolation mechanisms, such as virtualization and containerization techniques, to enable the application execution in sandboxes [18], or even unikernels [12]. Coupled with hardware extensions [19], these mechanisms can provide increased security for multi-tenant execution. These requirements of applications and resources introduce, from an algorithmic perspective, a high number of constraints that need to be addressed simultaneously while considering different optimization criteria.

III. Infrastructure Description

Our study focuses on a multi-layer edge-cloud infrastructure (Fig. 1), with computing and storage resources across various layers. The considered infrastructure comprises devices positioned at different locations, spanning from "near-edge" (i.e., from on-premises to tens of kilometres) to "far-edge" devices (i.e., some hundreds of kilometres) and cloud datacenters (i.e., typically several thousand kilometres away, situated in various geographic regions worldwide).

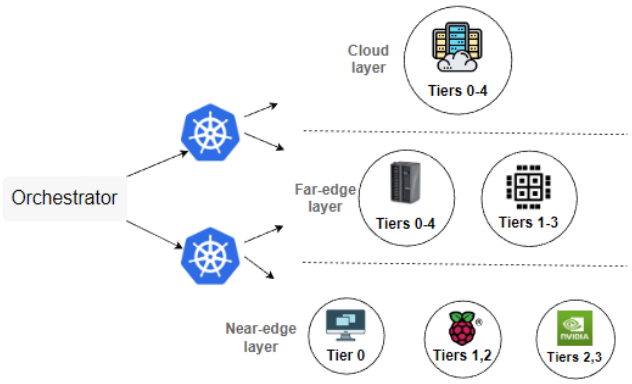

Fig. 1. Heterogenous resources across the edge-cloud continuum.

The management of the infrastructure and the service of the applications is enabled by a hierarchical two-level system. The high-level orchestrator assigns application requests to Local Orchestrators, based on well-established platforms (such as K8s, K3s) that control a subset of nodes in the infrastructure. This provides the freedom to the Local Orchestrator to serve in a highly efficient manner the "requests", satisfying both the resource's requirements and the central orchestrator objectives, with a minimal decision-making time. Next, the Local Orchestrators perform the actual

deployment based on the desired performance requirements. Hence, they perform the necessary resource allocation to serve the applications' microservices, optimizing a set of objectives.

The resources can vary both in size and capabilities, with the most common being micro-datacenters (mDC's), modular data centers in shipping containers, specialized computing devices (FPGA, GPU) and IoT devices (e.g., Arduino, Raspberry Pi, NVIDIA Jetson). These can be placed on providers' premises (e.g., the Central Office - CO), or on other large and small premises (e.g., stadiums, malls, businesses, houses) and special hardware enables the trusted execution.

Along the edge-cloud continuum, from IoT devices through the edge to the core cloud, available computing and storage capacity increases, culminating in an almost infinite resource pool in the cloud. The cloud provides abundant processing power and high availability, contrasted by edge resources' dynamic availability and less processing power. Execution cost pertains to expenses for operating computing/storage systems and special security-enhancing software/hardware. The cloud layer is typically the most cost-effective due to economies of scale, while cost increases for edge layers because of limited resources, a smaller customer base, hosting expenses, and wide geographical dispersion.

Various networking mechanisms using different wired (optical) and wireless (e.g., 5G) technologies provide the required interconnection of the individual edge and cloud layers. Typically, these multi-domain and multi-technology network paths are controlled and managed by multiple telco operators. In this work, we abstract the communication paths between the resources in the same or different layers as virtual links with specific latency and capacity. These values depend on the networking locality of the resources, with those in close proximity resulting in lower latency than those that are far apart. Hence, the propagation delay increases in accordance with the physical distance of the data-generation point.

To guarantee a robust and secure application execution, the infrastructure leverages advanced software mechanisms and in some cases peripheral employs specialized peripheral hardware. This comprehensive approach provides varying workload isolation levels, enabling also trusted execution across multiple layers, even in the presence of untrusted physical nodes commonly found in edge devices. This methodology aligns with the SERRANO H2020 project [6], which employs the confidential computing paradigm to create end-to-end security tiers. It transparently manages and spawns diverse containerized workloads, utilizing lightweight virtualization and strict security attestation mechanisms.

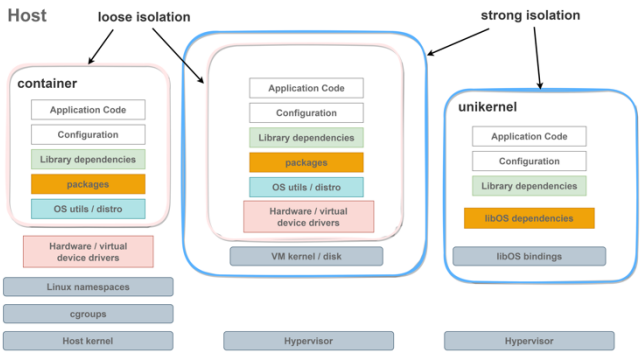

Fig. 2. Different levels of workload isolation provided by plain containers, unikernels and micro-VMs.

In Fig. 2, we show the diverse levels of workload isolation that can be achieved using lightweight virtualization mechanisms. These advancements pave the way for establishing distinct tiers of security and trustworthiness.

(i) *Tier-0* represents generic containers.

(ii) *Tier-1* embodies microVM sandboxing [10], [11]. In this scenario, the application, which is essentially the container, runs ontop of a microVM. This requires booting a complete virtualization stack (including VMM, kernel, rootfs), which remains active until the application's termination. Despite progress in minimizing the overhead of VMMs regarding CPU and memory footprint, even the latest VMMs [10] display a 30% overhead in memory management and address translation, plus additional CPU usage for handling I/O and context/mode switches. This also takes into account the extra memory used by the VMM and the necessity for a full OS system (the microVM) to be active for container spawning. Storage overhead is proportional to the application. However, a microVM can support container execution with a minimum rootfs, typically in the tens of MBs, with typical applications in the hundreds of MBs.

(iii) *Tier-2* is defined by unikernel execution. In this case, CPU, memory, and storage footprints are minimized as the application itself is compiled as a machine image, eliminating unnecessary OS and library software stacks. According to [19], this results in at least a 20% reduction in CPU and memory overhead, while the application binary footprint decreases by at least 60%. This reduction is achieved by excluding the OS/libraries from the application, apart from the optimized build.

(iv) *Tier-3* and (v) *Tier-4* are similar to Tier-0 and Tier-1 respectively, but with enhanced security provided by secure boot [18]. In these cases, a special peripheral hardware is required (known as Trusted Platform Module) to provide hardware based, security related functions. In addition, trusted execution in *Tier 4* requires the use of an attestation mechanism in the hypervisor layer.

Multipliers	Tier 0	Tier 1	Tier 2	Tier 3	Tier 4
CPU	1	1.3	0.8	1	1.3
RAM	1	1.3	0.8	1	1.3
Storage	1	1.1	0.4	1	1.1

Table 1. The multipliers of the computing and storage requirements for the different security and trustworthiness tiers.

Each tier imposes distinct demands in terms of computing and storage resources, which we have quantified in Table 1. The presented values are normalized with respect to the generic workload requirements of Tier 0. Hence, the multiplier of 1.3 of the CPU overhead for Tier 1 indicates that Tier 1 execution requires 30% more processing resources than Tier 0, while for Tier *2* 20% less. Hence, when deploying a cloud-native application, it is essential to provide: (i) the computing and storage requirements for each microservice, (ii) specify the maximum delay between them for optimal execution in the infrastructure and additionally, (iii) the minimum level of security and isolation for each microservice to ensure the application's secure and efficient operation.

IV. Prorblem Formulation

To represent a hierarchical edge-cloud infrastructure we assume a Complete Undirected Weighted Graph $G = (V, E)$. The set of nodes V corresponds to distinct geographical areas where a set M_v denotes the locations (nodes) where

computing resources are available and/or where workloads are generated (and can be equipped or not with local processing). A fixed communication (propagation) latency $l_{v,v'}$ is introduced among different nodes $v, v' \in V$. This latency takes into consideration the nodes' propagation delay, as well as additional delays incurred within the nodes during the communication process. Machines M_v (virtual and/or physical) are placed at the different nodes $v \in V$are controlled by low-level orchestrators O. Each low-level orchestrator $o \in O$ controls a subset of nodes $V_o \subseteq V$ and thus controls $M_o = \bigcup_{v \in V_o} M_v$ machines, with two orchestrators controlling distinct set of resources ($V_o \cap V_{o'} = \emptyset$, for $o, o' \in O$).

The machines serve the workloads at different security tiers $s \in S$, where integers are used to represent the different workload isolation levels from the lowest (equal to 0) to the highest. Also, a subset of the machines $V_s \subseteq V$ are equipped with hardware peripherals (secure boot) to support the execution of tier 3 and 4 workloads $S' = \{3,4\}$. Each machine m is described by the tuple $\tau_m = [c_m, r_m, h_m, t_m, p_m,]$, where c_m is the CPU capacity of the machine measured in CPU units, r_m is the RAM capacity of the machine measured in RAM units, h_m is the storage capacity of the machine measured in GB's, t_m indicates the existence of secure boot (value 1) or not (value 0) and p_m is the operational cost of the machine, which is the cost of use for a given period (time unit).

The workload in our scenario consists of a set A of cloud-native applications. Each application $a \in A$ is represented by an Undirected Weighted Graph $G^a = (V^a, E^a)$, where the nodes V^a denote the microservices that make up the application, and the edges E^a denote the existence of inter-dependencies among them. We adopted an undirected graph representation of the cloud-native applications, as we are concerned with the delay constraint formed by their communication dependency, which is assumed to be bi-directional. The data of each application is generated at node g_a. Each microservice $i_a \in V^a$, has specific requirements described by the tuple $[\varepsilon_{a,i}, \rho_{a,i}, \omega_{a,i}, \sigma_{a,i}, \lambda_{a,i}]$, where $\varepsilon_{a,i}$ is the microservice's CPU demand, $\rho_{a,i}$ is its memory demand, $\omega_{a,i}$ is the storage demand, $\sigma_{a,i}$ is the minimum security tier requirement and $\lambda_{a,i}$ is the duration of microservice in time units. Note that the computing and storage resources are specified assuming Tier *0* execution. This eliminates the need for users to profile the requirements of their applications for the different security tiers. Hence, when deploying the microservices in a machine with respect to the specified security tier requirement, the CPU, RAM and storage requirements of the microservices need to be considered based on the selected security tier s and thus with the respective multipliers $\hat{\varepsilon}_s, \hat{\rho}_s, \hat{\omega}_s$ (Table 1) to calculate the actual computing and storage requirements.

Moreover, each link e_{i_a,i'_a} that connects two microservices $i_\alpha, i'_a \in V^a$, with $i \neq i'$ denotes a maximum acceptable latency requirement δ_{i_a,i'_a}; this implies that microservices i_α, i'_a can be assigned to machines m, m' to corresponding service nodes v, v' only if $\delta_{i_a,i'_a} \geq l_{v,v':m \in v,m' \in v'}$. This delay constraint acts as a measure of the intensity of the dependency between them, in a sense that highly dependent microservices should be placed on the same or geographically approximate nodes. Finally, each application $a \in A$ has a delay limit D_a, which is the maximum acceptable delay between any node that hosts any of the application's microservices and the source node where the application's demand is generated. This is a general measure of the application's overall time-sensitivity, in the sense that a time sensitive application requires all its microservices to be processed by nodes with low-delay.

A. MILP FORMULATION

In what follows, we present the mathematical formulation of the two-level resource allocation problem of cloud native applications over an edge-cloud infrastructure. The optimization objective is the weighted combination of the total monetary cost, the delay per microservice assignment and the workloads' isolation level difference from the requested one. Below are the variables considered in the MILP formulation are presented.

$x_{a,i,o}$	Binary variable equal to 1 if microservice $i = 1, \dots, I_a$ of application $a = 1, \dots, A$ is assigned to low level orchestrator $o = 1, \dots, O$
$y_{a,i,m,s}$	Binary variable equal to 1 if microservice $i = 1, \dots, I_a$ of application $a = 1, \dots, A$ is placed at machine $m = 1, \dots, M_v$ and is served at security level s= $0, \dots, \lvert S \rvert$
θ_a	Integer variable that denotes the latency of application $a = 1, \dots, A$
Ψ_a	Integer variable that denotes the total monetary cost of serving the cloud native application $a = 1, \dots, A$
w_i	Weighting coefficients for $i = 1,2,3$ to control the contribution of operational cost and latency in the objective function with $\sum_{i=1}^{3} w_i = 1$

- Objective function.

$$\min w_1 \cdot \sum_{\alpha=1}^{A} \Psi_a + w_2 \sum_{\alpha=1}^{A} \theta_\alpha - w_3 \sum_{\alpha=1}^{A} \sum_{i=1}^{I_a} \sum_{m=1}^{M_o} \sum_{s=0}^{|S|} y_{a,\iota,m,s}$$

Subject to the following constraints:

-C.1. Each microservice $i = 1, \dots, I_a$ of each application $a = 1, \dots A$ must be assigned to a low-level orchestrator.

$$\forall a \in A, \forall i \in I_a, \sum_{o=1}^{O} x_{a,i,o} = 1$$

-C.2.The microservices $i = 1, \dots, I_a$ of each application $a = 1, \dots A$ that are executed with security Tier $s = 0, .., |S|$ must be assigned to a machine of the selected orchestrator *o*.

$$\forall a \in A, \forall i \in I_a, \forall o \in O, \forall m \in M_o, \sum_{m=1}^{M_o} \sum_{s=0}^{|S|} y_{a,\iota,m,s} \geq x_{a,i,o}$$

-C.3. The microservices that are executed with security Tier 3 and 4 need to be placed at nodes with extra peripheral hardware.

$$\forall a \in A, \forall i \in I_a, \forall o \in O, \forall m \in M_o, \forall s \in S', y_{i,a,m,s} \leq t_m$$

-C.4. The total CPU required from all the microservices $i = 1, \dots, I_a$ of application $a = 1, \dots, A$ deployed at a machine *m* must not exceed its capacity.

$$\forall o \in O, \forall m \in M_o, \sum_{a=1}^{A} \sum_{i=1}^{I_a} \sum_{s=1}^{|S|} (\varepsilon_{\iota,a} \cdot \hat{\varepsilon}_s) \cdot y_{a,i,m,s} \leq c_m$$

-C.5. The total RAM required from all the microservices $i = 1, \dots, I_a$ of application $a = 1, \dots, A$ deployed at a machine *m* must not exceed its capacity.

$$\forall o \in O, \forall m \in M_o, \sum_{a=1}^{A} \sum_{i=1}^{I_a} \sum_{s=1}^{|S|} (\rho_{\iota,a} \cdot \hat{\rho}_s) \cdot y_{a,i,m,s} \leq r_m$$

-C.6. The total Storage required from all the microservices $i = 1, \ldots, I_a$ of application $a = 1, \ldots, A$ deployed at a machine m must not exceed its capacity.

$$\forall o \in O, \forall m \in M_o, \sum_{a=1}^{A} \sum_{i=1}^{I_a} \sum_{s=1}^{|S|} (\omega_{\iota,a} \cdot \hat{\omega}_s) \cdot y_{a,i,m,s} \leq h_m$$

-C.7. The trusted execution tier of a machine that is assigned a microservice must be equal or greater than the tier demanded by the microservice.

$$\forall a \in A, \forall i \in I_a, \forall m \in M_o, \forall o \in O, \forall s \in S$$

$$y_{a,i,m,s} \cdot s \geq \sigma_{a,i}$$

-C.8,9. The microservices $i = 1, \ldots, I_a$ of application $a = 1, \ldots, A$ must be assigned to a machine that is situated in a node v that respects the application's delay limit.

$$\forall a \in A, \forall i \in I_a, \forall o \in O, \forall m \in M_o, \forall s \in S,$$

$$l_{m,g_a} \cdot y_{a,i,m,s} \leq \theta_a, \theta_a \leq D_a$$

-C.10. For each pair of connected microservices i,i' of an application $a = 1, \ldots, A$, the selected machines must respect the dependent microservices delay limit.

$$l_{v,v':m \in v, m' \in v'} \cdot y_{a,i,m,s} + l_{v,v':m \in v, m' \in v'} \cdot y_{a,i',m',s} \leq \delta_{a,i,a,i'} + l_{v,v':m \in v, m' \in v'}$$

-C.11 Monetary cost calculation for application $a = 1, \ldots, A$

$$\forall a \in A, \Psi_a = \sum_{i=1}^{I_a} \sum_{m=1}^{M_o} \sum_{s=0}^{|S|} y_{a,i,m,s} \cdot p_m \cdot \lambda_{a,i}$$

Assuming that A applications with I microservices need to be served. O orchestrators perform the assignment of applications to resources, with each one controlling M machines, which support S security tiers. The total number of variables is $[(A \cdot I \cdot O) + (A \cdot I \cdot M \cdot S) + 2 \cdot A + 3]$. The MILP formulation also requires $A \cdot (I + 1)$ equality constraints for C.1 and C.10. Also, it requires the following inequality constraints: $3 \cdot [A \cdot I \cdot M]$ for C.2,8,9, $2 \cdot [A \cdot I \cdot M \cdot S]$ for C.3.7, $3 \cdot M$ for C.4,5,6 and $A \cdot I^2$ for C.10.

B. Best fit heuristic

The presented MILP approach is computationally intensive and exhibits a prohibitively large execution time even for medium-sized problems. To address this, we developed sub-optimal mechanisms. The first mechanism is a greedy best fit heuristic. It takes as input the infrastructure graph G and application demands A, and allocates resources sequentially for the cloud native applications with respect to computing/storage capacity, security and latency needs optimizing the objective function set. To do so, it examines each microservice independently and allocates resources in a best-fit manner according to the specified objective function. When it fails to serve a microservice due to either communication latency or computing or storage capacity constraints it backtracks and re-allocates resources for the problematic set of microservices.

The algorithm begins by ordering the cloud native application demands based on their application delay limit D_a. As applications consist of dependent microservices, the pairs of microservices are also ordered based on their latency requirements (in latency units l.u.) among them from the strictest to the loosest. This way, the algorithm prioritizes applications and microservices with stricter latency requirements to maximize the chances of meeting the requirements while also decreasing any reallocations due to backtracking.

The allocation of resources for cloud native applications is performed sequentially. Given a microservice of an application a, the algorithm identifies the candidate orchestrators to serve it. These orchestrators are selected based on their ability to meet the application's latency requirement D_a and their machines' ability to fulfill communication constraints with already assigned microservices. Subsequently, the selected orchestrators are sorted in ascending order based on their objective value, which is the weighted average of their machines' cost, security, and latency towards the data generation node. The orchestrators are examined sequentially, beginning with the one offering the best objective value.

If an application contains only a single microservice, the algorithm selects the top-ranked orchestrator, and subsequently identifies candidate machines for deployment. These are machines that possess the required computing/storage resources and an equal or higher trusted execution tier than the one demanded by the microservice. Additionally, these machines must reside in nodes that satisfy the application's delay requirement. The algorithm then assigns the microservice to the candidate machine that yields the best objective value. If the application contains more than one microservices, the above process applies for the first microservice. However, for each subsequent microservice, the identification of the candidate machines also considers the latency requirements among interconnected microservices.

If no feasible placement is found for a microservice due to communication latency constraints, the affected inter-connected microservices that have already been served are de-allocated, freeing up the occupied resources. The algorithm will then attempt to re-embed the impacted microservices (possibly in a new orchestrator), until a feasible solution is found. This process is repeated until all microservices within an application are served, at which point the algorithm proceeds by selecting the next application in line. The algorithm terminates once all applications have been served.

C. Multi-agent Rollout heuristic

We also developed a multi-agent rollout [22] mechanism to enhance the performance of the greedy best fit heuristic and trade-off execution time with performance. Rollout is a well-known reinforcement learning technique that provide a near-optimal solution by leveraging a base policy, which in this case is the greedy best fit heuristic discussed in the previous sub-section. To find the final solutions, it follows an iterative process that takes, at each step, an instance of the problem with a partial solution and constructs the final solution incrementally. This approach is particularly useful when exact methods are slow or when the solutions provided by heuristics are below optimal and can produce significant results.

Upon selecting an application, the multi-agent rollout mechanism, illustrated in Fig. 3, begins by assigning an agent to each microservice. These agents co-operate/compete with time to fulfill the microservices' requirements based on the set objective function. Each agent takes sequential action, exploring all possible placements across the different orchestrators and their nodes. As the search space can be large, the algorithm prunes nodes that do not comprise machines meeting certain conditions. These conditions

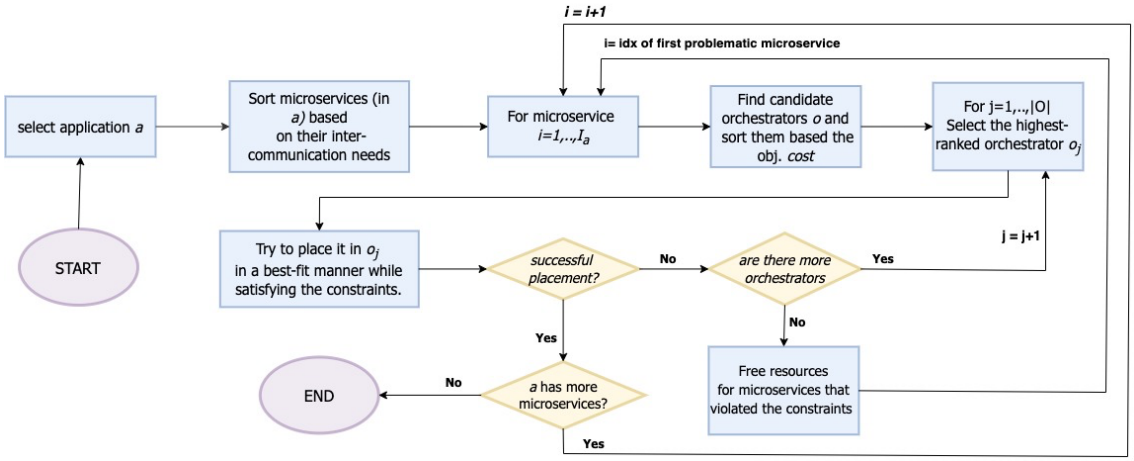

Fig. 3. The flowchart of the greedy best fit heuristic.

encompass: (i) the minimum latency requirements of the already served communicating microservices of the applications, (ii) the CPU, RAM and storage capacity. (iii) the minimum trusted execution requirements and (iv) the application latency constraint D_a. Furthermore, for each node, if more than one machines meet the problem's constraints, only the placement in the machine that yields the best objective is evaluated. This way, in the worst case, each agent assesses at most $\sum_o v_o$ possible placements for a microservice (instead of $\sum_o M_o$).

The best-fit heuristic of the previous sub-section is utilized by the Rollout mechanism in the context of reinforcement learning to approximate the future agents' decisions in the total cost approximation and evaluate the effect of the decision of the current agent. Hence, the process is repeated for all the candidate nodes that are able to serve the microservice of the current agent and the total cost that includes the current agent's decision and the approximate cost for future decisions is calculated and the assignment that exhibits the lowest total cost is selected in order to allocate resources for the current microservice. This methodology, thus, leads to a more informed and cost-effective approach to resource allocation, which can significantly improve the overall efficiency and performance of the cloud-native application deployment.

The purpose of using the multi-agent version of rollout is to curtail the state-space of the problem. By breaking down the allocation of resources for an application a and a microservice $i \in I_a$ and by applying one-agent-at-a-time instead of all-agents-at-once, the state space is effectively reduced. The reduction is further amplified by pruning the constraint-violating nodes, as well as evaluating only one machine per node, as discussed earlier. In this way, the control space complexity stemming from the various options for serving the applications is traded off with state space complexity, and the computational requirements are proportional to the number of microservices I_a of the different applications a and the number of nodes within the different orchestrators $\sum_o v_o$.

V. Performance Evaluation

A. Experimental setup

To examine the performance of the proposed mechanisms we performed a number of simulation experiments. The mechanisms were developed in MATLAB and the experiments were conducted on a 6 core 2.6 GHz Intel Core i7 PC with 12 GB of RAM. We assumed a hierarchical infrastructure that spans over the edge-cloud continuum and is split in three-layers that correspond to near edge, far edge and cloud nodes. We introduced two different topologies, namely "basic" and "extended", with each one consisting of nodes with computing machines of distinct characteristics and capacities, as illustrated in Table 2. Note that values exhibited in the close interval $[a, b]$ are sampled from the uniform distribution over that range.

	Near-edge	Far-edge	Cloud
Nodes (basic)	25	4	1
Nodes (extended)	40	7	2
Machines per node (basic)	1	[7,10]	50
Machines per node (extended)	2	[10,15]	100
CPU (CPU units)	[4,8]	[5,10]	[8,12]
RAM (RAM units)	[1,4]	[2,8]	[4,16]
STORAGE (GB units)	[4,16]	[8,32]	[16,64]
Monetary COST (Cost Units)	[6,7]	[3,4]	[1.5,2]

Table 2. The characteristics of the computing nodes of the different topologies.

In both topologies, the near-edge layer comprises of a large number of nodes with few low-capacity computing systems, placed close to the data sources. Conversely, the cloud layer comprises of a limited number of nodes that host a large number of high-powered machines. The cost of the near-edge resources, as compared to central cloud nodes, is assumed to be higher [23] as near-edge nodes are typically deployed in remote environments, making it challenging to maintain and upgrade the infrastructure. Also, these nodes are designed to have low latency and high bandwidth connectivity, which requires expensive networking equipment and bandwidth costs. Finally, the costs associated with providing the necessary power and cooling infrastructure is lower at the central cloud, which achieves economies of scale. Hence, the cost of the near-edge nodes was taken to be around 4 times higher than the central cloud, considering the additional cost associated with the secure workload execution.

We have also considered the communication delay between infrastructure nodes. We assumed that near-edge resources require between [0.5, 1.5] l.u., far-edge resources [3, 4] l.u. and cloud resources [7, 8] l.u. from the data generation points [24]. This takes into account the principle that nodes within the near-edge layer, given their geographic proximity

to the data source, should logically experience less propagation delay. Conversely, cloud resource nodes located in more distant areas would inherently experience longer delay times.

Number of microservices	[1,7]
Delay constraint	[2,10]
Microservices' CPU demand	[1,2]
Microservices' RAM demand	[0.5,1]
Microservices' storage demand	[1,5]
Dependency chance for a pair of microservices	25%
Dependency delay constraint	[0.5,3.5]

Table 3. The cloud native applications' workload characteristics.

For the workload, we focused on two scenarios; (i) a small and (ii) a medium-sized consisting of cloud-native applications of a maximum of 7 microservices (Table 3). Note that an application with a single microservice can represent a generic end-user demand, while microservice replicas are considered as microservices with identical resource profiles. We set the dependency probability between any pair of microservices to 25%, and the respective delay constraint to range between 0.5 and 3.5 l. u.

B. Evaluation Results

Initially, we compared the performance of the proposed sub-optimal mechanisms, the greedy heuristic and the multi-agent rollout with respect to the optimal solution provided by the MILP mechanism. For the evaluation we considered the following optimization criteria: (i) minimization of the operational cost ($w_1 = 1$), (ii) minimization of the applications latency ($w_2 = 1$), (iii) maximization of trusted execution ($w_3 = 1$) and (iv) all optimization criteria ($w_1 = w_2 = 0.4, w_3 = 0.2$). We used the "small" topology as described in Table 1 and a small workload consisting of 50 applications. The execution time for the optimal solver was limited to 60 minutes and the presented results are averaged over 20 simulations. The results of the simulation experiments are illustrated in Fig. 4.

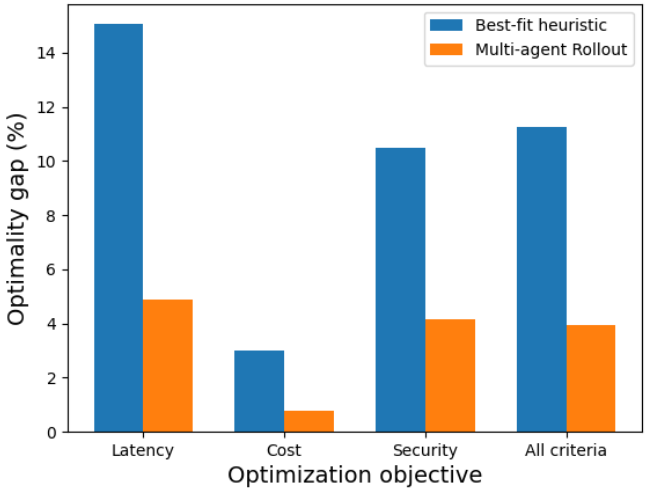

Fig. 4. The optimality gap for the different optimization criteria.

When the main optimization criterion is the latency minimization, the heuristic has an optimality gap of 14.5%. This happens due to the high competition for the limited near-edge resources, which requires a more sophisticated resource allocation approach to effectively allocate these resources. For the same reason the all-optimization criteria lags by 11% from optimal. However, when the optimization criterion is the minimization of the operation cost or the maximization of trusted execution, the search space is much smaller and thus the performance of the heuristic is close to optimal underperforming only by about 3% and 10% respectively.

On the other hand, the Multi-agent Rollout has a significantly better performance, with the worst case being for the latency optimization. However, it significantly improves the performance of the greedy heuristic due to the consideration of the future placements and thus lags only by 4.5%. This is the biggest gap for the performance of the rollout. Also, when the optimization is the minimization of the operational cost, the optimality gap is smaller than 1%, which means that it manages for the most of the application demands to allocate them in an optimal manner.

As for the execution time, the best fit heuristic provided an almost instantaneous assignment, with an average time of 0.01 seconds per application. On the other hand, the Rollout algorithm performed slower, at an average of 0.9 seconds per application, with a standard deviation of 0.3 seconds. Finally, the optimal solver exceeded the 3600 time-limit in all cases, thus resulting in an average of 72 seconds per application.

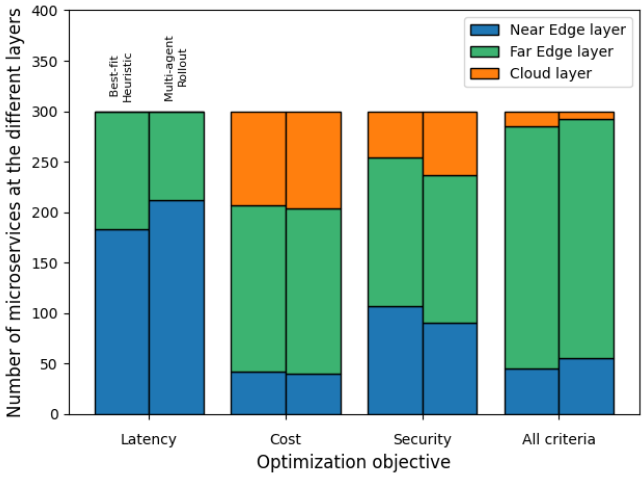

Fig. 5. The allocation of microservices at the different layers of the edge-cloud continuum.

Next, the multi-agent rollout mechanism was evaluated for the extended topology with 300 microservices and was compared to the allocation performed by the best-fit heuristic which is the baseline scenario for this set of experiments. We began by analysing the allocation of microservices for the different mechanisms and optimization criteria across the edge-cloud continuum (Fig. 5).

The experiments revealed that resource allocation patterns varied based on the optimization objective. When cost or the trusted execution were prioritized, cloud resources were favoured due to their high capacity and the higher availability of trusted execution tiers. Conversely, when the latency minimization was the main objective, near and far edge resources were heavily utilized. Additionally, when all the optimization criteria were simultaneously optimized, the solution proved beneficial in allocating resources tailored to the application's specific needs.

This highlights the advantages of taking all the optimization criteria into consideration in a multi-objective optimization approach during the resource allocation process and the ability of the rollout mechanism to achieve an improved allocation of resources by leveraging the decisions of the heuristic in a reinforcement learning manner. All in all, this approach leads to more efficient allocation of resources

across the edge-cloud continuum, optimizing application performance and cost.

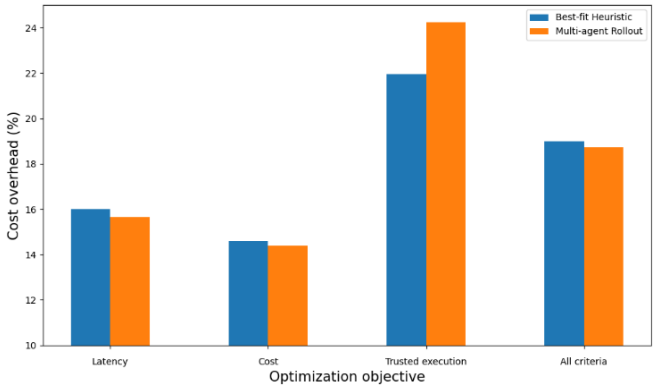

Fig. 6. The operational cost overhead for the different optimization criteria.

Next, in Fig. 6 we present the results of experiments regarding the average cost overhead associated with security as an additional constraint, compared to generic workload demands which acts as the baseline scenario for this case, for the different optimization objectives. The cost overhead for a microservice's placement is determined as the percentile increase in cost between its deployment in a default container (Tier-0) and the deployment method chosen in the assignment.

As expected, the highest cost overhead is incurred in the maximization of trusted execution, where machines with higher tiers, which are inherently more costly, are favoured. Trusted execution is also considered in the "all optimization criteria" scenario, producing increased cost overhead. On the other hand, the impact of additional security on cost overhead is less notable for the other two optimization objectives, where trusted execution is not a contributing factor.

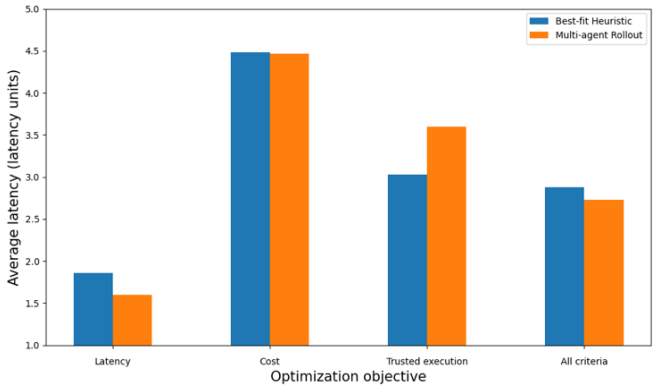

Fig. 7. The experienced latency for the different optimization criteria.

Finally, in Fig. 7 we present the effect of the different optimization criteria on the average propagation latency. When the minimization of latency is prioritized, the lowest average latency is achieved. Conversely, when operational cost is optimized, as many microservices as possible are placed on the far-edge and cloud layers, incurring higher latency. In the trusted execution optimization case, microservices are placed on machines with high security tiers, which are generally located in the upper layers (extra hardware), leading to higher average propagation latency. The weighted optimization approach strikes a balance among the different criteria, thus achieving a relatively smaller latency.

By comparing the Multi-agent Rollout with the greedy heuristic mechanism performance, we observe that the rollout approach generally results in a marginally lower cost overhead and communication latency for all objectives, with the exception of trusted execution. As expected, when considering both cost and other criteria, the rollout mechanism performs better, finding the most cost-efficient machines that usually possess lower security levels and subsequently lower security-cost-overhead.

Similarly, with latency as an objective, it manages to place more microservices on edge nodes, at machines with adequate security tiers and thus lower overhead costs. Similarly, for the trusted execution objective, the rollout mechanism improves the solution by placing more microservices on machines with a higher security tier, leading to a higher security cost overhead. The aforementioned findings highlight the importance of considering multiple optimization criteria when allocating resources for cloud native applications across the edge-cloud continuum. While prioritizing a single objective may lead to optimal results for that specific objective, it may negatively impact other criteria, such as latency or operational cost. Therefore, a comprehensive approach that balances multiple objectives can lead to a more efficient allocation of resources, resulting in improved application performance, reduced costs and better utilization of the infrastructure's resources.

VI. Conclusion

In this study, we aimed to address the challenge of allocating resources to cloud-native applications within a hierarchical edge-cloud infrastructure. Our approach considered critical factors such as the inter-dependencies among microservices and the trusted execution requirements of cloud-native applications. To meet the varied security and isolation demands of microservices, we introduced innovative technologies such as sandboxing and unikernels into our infrastructure. To model the resource allocation problem, we formulated a multi-objective optimization problem that balances various and conflicting objectives, such as minimizing operational costs and propagation latency from data generation points, while considering the workloads' security tier requirements. We developed optimal and sub-optimal mechanisms that efficiently trade-off performance for execution time, as demonstrated in our experiments. Our results showed that the greedy best-fit heuristic fell short of optimal performance by almost 11% for all optimization criteria. However, the multi-agent rollout mechanism significantly improved the greedy heuristic's performance, achieving close to optimal levels at 3.7%. Furthermore, our experiments highlighted the trade-offs between delay, cost, and security. In conclusion, our study provides a novel approach to resource allocation in a hierarchical edge-cloud infrastructure, addressing crucial factors such as security, isolation, and inter-dependencies among microservices. The proposed multi-objective optimization problem and developed mechanisms offer efficient trade-offs between performance and execution time.

Acknowledgment

The work presented is supported by the EU Horizon 2020 research and innovation program under grant agreement No. 101017168 in the context of the SERRANO project and by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 04596).

REFERENCES

[1] Y. bin Zikria, R. Ali, M. K. Afzal, and S. W. Kim, "Next-generation internet of things (IoT): Opportunities, challenges, and solutions," Sensors (Switzerland), vol. 21, no. 4. MDPI AG, pp. 1–7, Feb. 02, 2021.

[2] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of 'big data' on cloud computing: Review and open research issues," Information Systems, vol. 47. Elsevier Ltd, pp. 98–115, 2015.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," IEEE Internet Things J, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] G. Sallam and B. Ji, "Joint Placement and Allocation of VNF Nodes with Budget and Capacity Constraints," Jan. 2019.

[5] J. Singh, J. Powles, T. Pasquier, and J. Bacon, "Data flow management and compliance in cloud computing," IEEE Cloud Computing, vol. 2, no. 4, pp. 24–32, Jul. 2015.

[6] "SERRANO - Transparent Application Deployment in a Secure, Accelerated and Cognitive Cloud Continuum" https://ict-serrano.eu (accessed Jun. 26, 2023).

[7] X. Li, Z. Lian, X. Qin, and W. Jie, "Topology-aware resource allocation for IoT services in clouds," *IEEE Access*, vol. 6, pp. 77880–77889, 2018.

[8] R. A. C. da Silva and N. L. S. da Fonseca, "Resource allocation mechanism for a fog-cloud infrastructure," in *IEEE International Conference on Communications*, Jul. 2018, vol. 2018-May.

[9] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini and S. Cretti, "Foggy: A Platform for Workload Orchestration in a Fog Computing Environment," 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, China, 2017.

[10] A. Alexandru, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, D. M. Popa. "Firecracker: Lightweight Virtualization for Serverless Applications." In NSDI, vol. 20, pp. 419-434. 2020.

[11] Kata Containers, "The speed of containers, the security of VMs," Available online: https://katacontainers.io/

[12] A. Madhavapeddy et al., "Unikernels," ACM SIGARCH Computer Architecture News, vol. 41, no. 1, pp. 461–472, 2013.

[13] Xiong and H. Chen, "Challenges for Building a Cloud Native Scalable and Trustable Multi-tenant AIoT Platform," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, Nov. 2020.

[14] S. Meng *et al.*, "Security-Aware Dynamic Scheduling for Real-Time Optimization in Cloud-Based Industrial Applications," *IEEE Trans Industr Inform*, vol. 17, no. 6, pp. 4219–4228, Jun. 2021.

[15] Y. Wang, W. Zhang, H. Deng, and X. Li, "Efficient Resource Allocation for Security-Aware Task Offloading in MEC System Using DVS," *Electronics (Switzerland)*, vol. 11, no. 19, Oct. 2022.

[16] Z. Li, V. Chang, H. Hu, D. Yu, J. Ge, and B. Huang, "Profit maximization for security-aware task offloading in edge-cloud environment," *J Parallel Distrib Comput*, vol. 157, pp. 43–55, Nov. 2021.

[17] S. Shen, T. Zhu, D. Wu, W. Wang, and W. Zhou, "From Distributed Machine Learning To Federated Learning: In The View Of Data Privacy And Security," Oct. 2020.

[18] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 2015, pp. 57-64

[19] S. Kuenzer et al., "Unikraft: Fast, Specialized Unikernels the Easy Way," arXiv.org, Apr. 21, 2021.

[20] Nicholas, G.S., Siddiqui, A.S., Joseph, S.R. et al. A Secure Boot Framework with Multi-security Features and Logic-Locking Applications for Reconfigurable Logic. J Hardw Syst Secur 5, 260–268, 2021.

[21] R. Tarjan, "Depth-first search and linear graph algorithms," 12th Annual Symposium on Switching and Automata Theory, pp. 114-121, USA, 1971.

[22] D. Bertsekas, "Multiagent Reinforcement Learning: Rollout and Policy Iteration," in IEEE/CAA Journal of Automatica Sinica, vol. 8, no. 2, pp. 249-272, 2021.

[23] "Analysis: The economics of edge computing." Accessed: Mar. 06, 2023. [Online]. Available: https://www.edgecomputing-news.com/2020/10/29/analysis-economics-of-edge-computing/

[24] Pallewatta, S., Kostakos, V., & Buyya, R., "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments", UCC 2019 - Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing,71–81.