Multi-Criteria Optimization of Application Offloading in the Edge-to-Cloud Continuum

Branko Miloradović and Alessandro Vittorio Papadopoulos

Abstract— **Applications are becoming increasingly data-intensive, requiring significant computational resources to meet their demand. Cloud-based services are insufficient to meet such demand, leading to a shift of the computation towards the devices closer to the edge of the network, leading to the emergence of an Edge-to-Cloud computing Continuum (E2C). An application can offload part of its computation toward the E2C. The allocation of applications to a set of available computing nodes is a challenging problem, as the allocation needs to take into account several factors, including the application requirements and demands as well as the optimization of the resource utilization in the E2C infrastructure and the minimization the CO_2 footprint of the executed applications. Control and optimization techniques provide a vast array of tools for optimizing the Edge-to-Cloud continuum's management. This paper provides a mathematical formulation for the application offloading with specific requirements in the cloud computing domain. The problem is modeled as integer linear programming and constraint programming models and implemented in commercially available software. Finally, we provide the results of performed comparison between the two models.**

I. Introduction

The number of connected devices that extensively use data is increasing exponentially, and Internet of Things (IoT) devices are expected to reach 34.7 billion by 2028 [2]. To support such an increasing demand for data-intensive application processing, new computational models have been developed, building and extending the cloud computing paradigm. In particular, in the last few years, we are experiencing a shift of the computation toward the devices closer to the edge of the network, leading to the emergence of an Edge-to-Cloud computing Continuum (E2C) [12], [6]. Application offloading from end devices to the Edge-to-Cloud (E2C) continuum became a necessity [31], [10], especially when considering advanced and distributed control [27], [26], [28], [7] or Machine Learning [19] applications. Different applications have different requirements in terms of low latency and real-time responses. In conjunction with large-scale geographical distribution and heterogeneity of E2C computational nodes, these requirements make application offloading in such infrastructure a challenging issue [32]. Diversity of user expectations and IoT device characteristics also complexify the offloading problem. In addition, cloud providers often have limited resources, such as computing power, storage capacity, and network bandwidth, and need to allocate these resources to satisfy the application demands optimally.

In an ideal scenario, all end devices can perform the necessary computation without offloading the work to another system. Since that is not often possible, the computational offloading approach transfers the computational load to a device capable of performing such work. Most research assumes a homogeneous set of nodes [32], [10] in terms of architectures. Still, the advent of new computing architectures, such as Graphical Processing Units (GPUs), Tensor Processing Units (TPUs), and quantum technologies, does not allow for the seamless execution of applications on any generic node, thus calling for more flexible approaches. In this paper, we propose an approach that allows an application to require specific capabilities from the nodes where it runs, e.g., access to a camera or sensors, the possibility of performing AI computations on GPUs, etc.

What complicates offloading is the latency requirement of applications, i.e., some applications have time-critical components, and the latency constraint is a factor that must be considered when performing an offloading operation. This is the typical case for control applications that require strict timing to guarantee certain Quality of Control metrics [31].

These are the requirements from the device (user); on the other hand, we have requirements from the service providers. In recent years, there has been a lot of attention towards the green transition, and E2C infrastructures can make use of so-called "green nodes" [3], i.e., nodes that are powered by green energy in contrast to fossil-fueled energy. This means that whenever possible, the usage of such nodes should be a priority to reduce the carbon footprint [19]. In addition, the requirement is to increase the utilization of the nodes in general, which would minimize the total number of nodes used, lowering the overall cost. This problem can be seen as a generalization of the bin packing problem, which is NP-hard. In this paper, we introduce a multi-criteria optimization approach to account for the several stakeholders involved in the application offloading to capture the applications' needs and requirements while minimizing the usage and environmental impact of the E2C infrastructure.

To this end, we proposed a mathematical model to formally describe the problem, which has been implemented and verified in two commercially available solvers. In particular, the contribution of this paper is threefold:

- a mathematical model of the offline application offloading problem;
- implementation of the mathematical model both as an Integer Linear Constraint (ILP) model in CPLEX and as

This work was supported by the Swedish Research Council (VR) with the PSI project (No. #2020-05094), by the Knowledge Foundation (KKS) with the FIESTA project (No. #20190034).

B. Miloradović and A.V. Papadopoulos are with Mälardalen University, Sweden {name.surname}@mdu.se

a Constraint Programming (CP) model in CP Optimizer;
- analysis of the performance of two different solvers on a series of test instances.

II. Related Work

In recent years, there have been numerous survey papers on computation offloading in the cloud [1], [14], edge computing [20], [25], [33], taxonomy and open issues [16]. There have been so many surveys in the past couple of years that we have a study on surveys [8] regarding computation offloading in the E2C continuum. The said paper also provides a survey on optimization techniques used for offloading, which will be the scope of this section. More precisely, following the taxonomy described in [8], we will be focusing on *Device - Edge-to-Cloud hybrid offloading*, as the offloading can be done in both vertical and horizontal manner, depending on the network architecture. In particular, we allow the horizontal offloading to be performed as a federation of upward and downward offloading (e.g., from the edge node to the cloud node and down to a different edge node) in cases when there is no direct link between the nodes in the same level of abstraction.

There are different techniques used for application offloading optimization. However, the most prominent is Machine Learning (ML) [21], traditional optimization techniques that include ILP [9], different suboptimal heuristics [11], [15], and optimal algorithms [5], [13]. Mouradian *et al.* [17] provide a survey on optimization techniques in cloud-based offloading.

In the literature, application placement or service allocation has also been used to refer to the offloading of jobs from end devices to a node in the cloud domain. Recent surveys covering this topic have been done by Smolka and Zolt [29] and Aït-Salaht *et al.* [22]. Souze *et al.* [30] present a paper on service allocation in fog-cloud scenarios. They modeled service allocation as an ILP problem and use off-the-shelf solvers PuLP and Gurobi to optimize a problem consisting of 30–90 services. Nishio *et al.* [18] present heterogeneous resource sharing for optimizing service latency in the cloud. The problem is a convex optimization problem that maximizes utilization and minimizes the system's latency. Finally, Aït-Salaht *et al.* [23] model the computation offloading problem as a Constraint Programming (CP) problem.

However, most of the proposed solutions do not consider the complexity emerging from the heterogeneity of technologies and application requirements. They usually assume homogeneous infrastructures, where an application can be offloaded on any computing node with enough cores and memory. In this paper, we propose a model that can account for (i) a heterogeneous set of nodes, such as different platforms and technologies, e.g., CPUs, GPUs, and TPUs, (ii) the presence of green nodes in the infrastructure that can be used to minimize the carbon footprint of the computation, and (iii) multiple objectives that can influence the placement, such as the latency of an application, the utilization of nodes, and the environmental impact of the computation offloading.

III. Application Offloading Problem: Model and Formulation

In this section, we will describe, in detail, the model and the problem formulation of application offloading in the Edge-to-Cloud Continuum. Before describing the optimization problem, the system model will be presented.

A. E2C model

The system can be divided into two subgroups. The first one is the infrastructure of the Edge-to-Cloud model, where the applications are deployed. The second is the application requirements.

1) Infrastructure: The E2C model is an undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, consisting of a set of n nodes $\mathcal{N}$ and a set of e edges $\mathcal{E}$. The nodes represent the processing/computational components of the E2C infrastructure, and the edges represent possible communication routes between the nodes. Depending on their hierarchical level, the nodes can be further segregated into edge, fog, and cloud nodes. These types of nodes mainly differ in the capacity to handle offloaded applications. For example, edge nodes are envisioned as resources that provide computational power in the range of routers, mobile phones, tablets, IoT devices, etc. Fog nodes have more computational capacity and may include different types of personal computers. Finally, cloud nodes are data centers. The other difference between edge nodes on one side and fog and cloud nodes on the other is that the applications are always first communicated to edge nodes. The part of the optimization problem is then to decide if and where to offload the application, depending on the objective function. A certain metric characterizes each node. These metrics include the number of cores, available RAM, special capability (e.g., certain sensors, ability to perform GPU computations, the required level of security, etc.), and the type of energy they use for running. To simplify things,

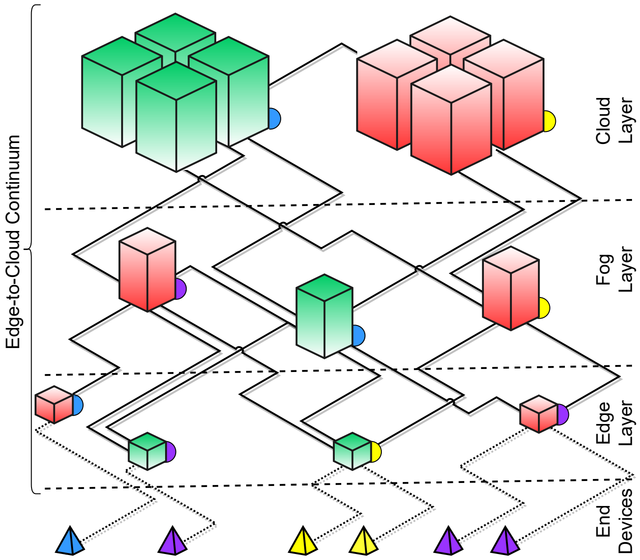

Fig. 1. E2C infrastructure overview.

all nodes are binary in this metric, meaning we assume a node is either fully running on green energy or fossil fuels.

2) Applications: Applications, similarly to nodes, are also characterized by certain metrics. Each application has a requirement on the number of nodes, size of RAM, capability a node needs to possess, maximum latency allowed, and information on which edge node it was initially offloaded. It should be noted that it is assumed that applications are atomic, i.e., an application cannot be decomposed and offloaded to multiple nodes.

In Fig. 1, an infrastructure of the E2C model is presented. The polyhedrons represent the nodes in the system, while their color shows if they are running on green energy. They also have a half circle attached with different colors. These colors represent the special equipment the node posses. The tetrahedrons at the bottom are applications to be offloaded. Their color should match the color of a half circle of the nodes to be able to be offloaded to that node. The lines connecting different parts of the E2C are possible communication routes.

B. Problem description and Terminology

In this work, offloading is defined as transferring the execution of applications from a system that cannot perform them to nodes in the E2C continuum that can. In other words, offloading is a mapping between the set of applications $\mathcal{A}$ and a set of node $\mathcal{N}$ in the E2C infrastructure $\mathcal{G}$. The graph $\mathcal{G}$ is assumed to be a connected graph, and this allows for application offloading to any node in the network, independently of the 1-hop connection that the application establishes with the E2C infrastructure. The link $e_{i,j}$ connecting nodes i and j is associated with a latency metric, and we can calculate the latency between any two nodes in the network. In this work, we used the Dijkstra algorithm to calculate the shortest path (latency) between every pair of nodes and populate the cumulative latency matrix $\Omega = [\omega_{i,j}]_{n \times n}$, however, other algorithms can be used in this context, e.g., A*, Bellman-Ford, Floyd–Warshall, etc. It is assumed that the network topology is known *a priori*; thus, it is enough to do this calculation only once. The complexity of the Dijkstra algorithm using binary heap implementation is $\mathrm{O}((n+e)\log n)$ for a pair of nodes. In this work, we do not take link bandwidth into account, as the bandwidth is not considered the limiting resource in this work problem. We focus on the special capabilities of nodes to process certain requests from offloaded applications, which are not necessarily bandwidth-demanding.

For example, a mobile application can offload an image processing task to a cloud-based service if it does not have enough computation power to perform it in a reasonable amount of time. The offloading, in return, entails a certain latency, which we take into account in this paper. We assume that in the system, latency between nodes is known for each pair of nodes.

The problem addressed in this paper is the offloading of a set of applications $\mathcal{A} := \{a_1, a_2, \ldots, a_m\}$ to a set of nodes $\mathcal{N}$. This set consists of three subsets; the subset $\mathcal{D}$ consisting of p number of edge nodes; the subset $\mathcal{F}$ consisting of q number of fog nodes; and the subset $\mathcal{C}$ consisting of w number of cloud nodes. The set containing all nodes in the network is defined as $\mathcal{N} := \mathcal{D} \cup \mathcal{F} \cup \mathcal{C}$. The total number of nodes in the network is $n = p + q + w$. Each node has a location, and the set denoting all nodes' locations is $\mathcal{L}$.

Since the network is not only heterogeneous in terms of computation power but also in terms of special equipment, we can also define a set containing k number of different equipment in the system as $\Xi := \{\xi_1, \xi_2, \ldots, \xi_k\}$.

a) Definition 1 (Node): A node $n \in \mathcal{N}$ is a collection $n = \langle c_n, m_n, \xi_n, \gamma_n, l_n \rangle$, where

- $c_n \in \mathbb{Z}^+$ is the number of cores available on the node n.
- $m_n \in \mathbb{Z}^+$ is the amount of RAM memory available on the node n.
- $\kappa_{\text{node}}(n) : \mathcal{N} \mapsto \Xi$ is a function that gives the set of equipment elements ξ_i available on a node n.
- $\gamma_n \in \{0, 1\}$ denotes whether the node is running on green energy.
- $l_n \in \mathcal{L}$ is the location of node n.

b) Definition 2 (Application): An application $a \in \mathcal{A}$ is a collection $a = \langle c_a, m_a, \xi_a, l_a, \ell_a \rangle$, where

- $c_a \in \mathbb{Z}^+$ is the number of cores required by the application a.
- $m_a \in \mathbb{Z}^+$ is the amount of RAM memory required by application a.
- $\kappa_{\text{app}}(a) : \mathcal{A} \mapsto \Xi$ is a function that gives the set of equipment elements ξ_i necessary to run a.
- $l_a \in \mathcal{L}^{\mathcal{D}}$ is the node location where the application has been initially offloaded, and $\mathcal{L}^{\mathcal{D}}$ is the set containing locations of edge nodes $\mathcal{D} \subseteq \mathcal{N}$.
- $\ell_a \in \mathbb{Z}^+$ denotes the maximum allowed latency by application a.

For the sake of simplicity and without loss of generality, we assume that the frequency of all cores is identical. The same applies to the RAM.

C. Problem formulation

This section presents the problem formulation of application offloading in the E2C continuum using the set of variables, their domains, and a set of constraints.

First, we define the following:

- $\delta : \mathcal{A} \mapsto \mathcal{N}$ is a function that maps the applications to the nodes to which they are offloaded. Each application is mapped to exactly one node.
- $\eta : \mathcal{N} \mapsto \mathbb{Z}^+$ is a function that gives the number of utilized cores on a given node. It is upper bounded by the number of cores on the considered node n, i.e.,

$$\eta(n) \leq c_n, \quad \forall n \in \mathcal{N}$$

- $\mu : \mathcal{N} \mapsto \mathbb{Z}^+$ is a function that gives the amount of memory utilized on a given node. It is upper bounded by the amount of memory on node n, i.e.,

$$\mu(n) \leq m_n, \quad \forall n \in \mathcal{N}$$

- $\nu : \mathcal{N} \mapsto \{0,1\}$ represents if a node is used or not and is defined as

$$\nu(n) = \begin{cases} 1, & \text{if } \eta(n) > 0 \vee \mu(n) > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

In Eq. (1), we make the connection between variables $\eta(n)$ and $\nu(n)$ such that whenever a node n is in use, its core usage has to be larger than 0. This constraint would also work on memory instead of the core since the lower bound on the application requirement for cores and memory is never smaller than 1.

Now that decision variables have been defined, we can introduce constraints. The first constraint type we are going to use is known as bin packing [24]. In our work, the bin packing constraint ensures that the physical limitations of nodes are not exceeded during the application offloading process. Let $\mathcal{A}_n = \{a \in \mathcal{A} | \delta(a) = n\}$, then the sum of the required number of cores c_a and memory m_a of all applications $a \in \mathcal{A}_n$ offloaded to node n must not exceed the node core and memory capacity, i.e.,

$$\sum_{a \in \mathcal{A}_n} c_a \leq c_n, \quad \forall n \in \mathcal{N}, \tag{2}$$

$$\sum_{a \in \mathcal{A}_n} m_a \leq m_n, \quad \forall n \in \mathcal{N}, \tag{3}$$

where $\delta(a)$ represents the node where the application a is offloaded. To make sure that the selected node for offloading has the required equipment required by the offloaded application, we introduce the following constraint

$$\kappa_{\text{app}}(a) \subseteq \kappa_{\text{node}}(\delta(a)), \quad \forall a \in \mathcal{A}. \tag{4}$$

Finally, we formulate the constraint on the latency requirements of applications. If an application a has a requirement on the maximum latency ℓ_a that the application must experience, then

$$\omega_{l_a,\delta(a)} \leq \ell_a, \quad \forall a \in \mathcal{A}. \tag{5}$$

Now that the decision variables and constraints have been introduced and explained, defining the optimization criteria is the only missing part of the problem formulation. In these types of problems, there can be many different optimization criteria [22]. In this work, we focus on three different criteria. The first one is the minimization of the use of fossil-fueled nodes. The second one is to minimize the overall latency in the system. Finally, the third criterion minimizes the total number of nodes used. This specific order of importance is chosen because our goal is to minimize the carbon footprint while keeping a good response time in the system.

$$f_1 = \sum_{i \in \mathcal{N}} \gamma_i \cdot \nu(i). \tag{6}$$

$$f_2 = \sum_{i \in \mathcal{A}} \omega_{l_i,\delta(i)} \cdot \nu_{\delta(i)}. \tag{7}$$

$$f_3 = \sum_{i \in \mathcal{N}} \nu(i). \tag{8}$$

The way we combine these criteria is lexicographic optimization [4], i.e., their ranking is in the order they are mentioned in the text.

$$\begin{aligned} &\text{lex} \min_{\delta,\eta,\mu,\nu} && f_1, f_2, f_3 \\ &\text{subject to} && \text{Constraints (1)–(5)} \end{aligned}$$

This concludes the problem formulation. In the next section, we will present the results of our benchmarks.

IV. Results

This section presents the evaluation of the proposed formulation. After describing the used experimental setup, we present a benchmark of different problem instances used in the assessment of the proposed method. The experimental platform is an i9-9980XE @3.8GHz (18 cores) CPU with 128 GB of DDR4 RAM.

A. Benchmark

The benchmark consists of 10 randomly generated test instances, shown in Table I. The table shows the number of edge, fog, and cloud nodes, the number of green nodes, the percentage of green nodes out of total nodes, and the total number of nodes and applications to be offloaded. The instances gradually increase in the number of nodes and applications to assess the scalability of the considered approaches. All of the test instances have feasible solutions. As defined in problem formulation (Sect. III-C), attributes of nodes are the number of cores available, amount of RAM, special equipment available, and energy type used to run the node. When creating the test instances, we bounded the randomness of the variables. More specifically, edge nodes can have 1-8 cores and 4-32 GB of RAM. This is increased to 8-32 cores and 32-128 GB of memory for fog nodes. Finally, cloud nodes have 32-128 cores and 256-512 GB of memory. Nodes can have none or one of three specialized equipment. Finally, a node can be run by green energy or fossil fuels. All of these values are acquired randomly, using a uniform distribution.

Applications have attributes such as the number of cores required, the amount of RAM required, special equipment required, and the location where a 1-hop connection was first established with E2C infrastructure. As is the case with nodes, application variables are also bound. Specifically, the number of cores required is 1-8, and the amount of memory

TABLE I
INFORMATION ABOUT TEST INSTANCES USED IN THE BENCHMARK.

Inst.	# Edge	# Fog	# Cloud	Total	# Green	Green/Total	# Apps
1	4	4	2	10	6	60 %	25
2	8	8	4	20	11	55 %	50
3	12	12	6	30	12	40 %	100
4	18	18	9	45	23	27 %	150
5	22	22	11	55	29	53 %	200
6	24	24	12	60	32	53 %	300
7	26	26	13	65	33	51 %	400
8	34	51	17	85	51	60 %	450
9	36	36	18	90	47	52 %	500
10	48	48	24	120	58	48 %	600

is between 1-8 GB. There is a 40 % chance an app will need one of 3 special pieces of equipment. The E2C node used in 1-hop connections is chosen randomly from the set of edge nodes. Finally, there is a 10% chance that an application will have a latency constraint, i.e., an upper bound.

The full set of test instances we created for this benchmark can be found online[1]. The benchmark consists of solving 10 test instances generated in the previously described way and comparing the results gathered from the two solvers used. The first solver is CPLEX, used for solving ILP problems. The second one is the CP optimizer, used for constraint programming problems. We compare the results for each optimization criterion and the time taken to obtain those results. The search time limit is set to 1 hour for both solvers.

B. CP Solver

Constraint programming involves modeling the problem in terms of variables, domains, and constraints and then searching for a solution that satisfies all the constraints. It has applications in various fields, including scheduling, planning, optimization, and artificial intelligence. The key advantages of CP include its ability to handle complex problems with many interrelated constraints, find optimal or near-optimal solutions, and its flexibility in allowing for different types of constraints and objectives. There are many off-the-shelf solvers available to solve CP models, e.g., MiniZinc, Google's OR-Tools, IBM's CP optimizer, etc. The goal is not to compare the performance of different solvers, any of these solvers is a reasonably good choice, and our choice in this work is IBM's CP Optimizer. Regarding the optimization parameters, we use `IloSearchPhase` to specify which decision variable the solver should instantiate first in the search process. For our problem, we instantiate first the vector of decision variable $\{\delta(a_1), \ldots, \delta(a_m)\}$ as it has been shown that in similar problems, fixing the allocation part first yields better overall results. This has been proven to be the case here as well.

C. ILP Solver

As is the case with CP solvers, many ILP solvers are available. We decided to use the one also developed by IBM called CPLEX. It is a state-of-the-art optimization tool capable of solving various ILP problems. As a tool that has been under development for decades, it incorporates a wide selection of algorithms with tunable parameters. In this work, we use CPLEX with its default settings.

D. Performance Analysis

Before moving to performance analysis and benchmarks, we will first give an example of a simple application offloading problem that consists of 5 nodes and 7 applications (Fig. 2). Nodes are enumerated from 1 to 5, and for the sake of this example, let's assume that nodes 1 and 2 are in the edge layer, nodes 3 and 4 are in the fog layer, and node 5 is in the cloud layer. Out of 7 applications, 1-3 require GPU computing, 4-5 need a camera, and 6-7 have a high-security level requirement. For the sake of simplicity, we did not put explicit information on the core/memory in the figure. For the same reason, latency is excluded from the figure. The 1-hop connection for applications 1-4 to E2C is node 1, and for applications, 5-7 is node 2. Since nodes 1 and 2 are edge nodes, they are assumed to have low computational capacity. Therefore, application 4 is allocated to node 1 and application 5 to node 2, since neither can perform both applications. If, for example, node 2 could, then application 4 would be allocated to node 1 through other nodes in the network since we do not have a direct link between nodes 1 and 2. Node 1 cannot perform applications 1-3, which are further offloaded to fog node 3. Node 3 does not have enough computation power for all three applications, and 1 and 2 are further offloaded to cloud node 5. Both nodes 3 and 5 are powered by green energy, as indicated by a green symbol. Applications 6 and 7 require a high level of security, which only node 5 posses, and their 1-hop connection to E2C is through node 2, so they are offloaded to node 3 through node 2. Each link has a latency metric assigned to it. This was just a simple example of the application offloading. Now we will focus on the performance analysis of solvers solving test instances of a size up to 120 nodes and 600 applications.

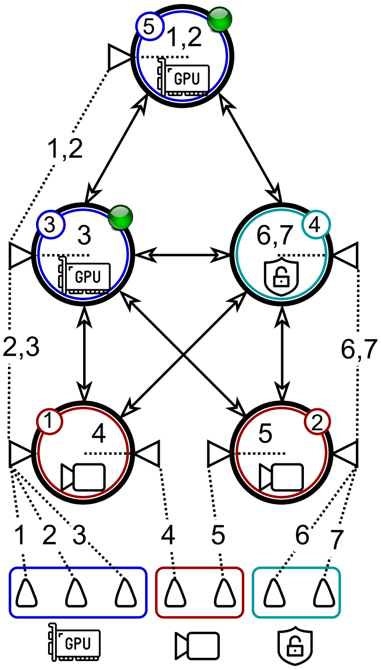

Fig. 2. An example of a solution to a simple application offloading problem. Nodes are represented as circles, while applications have a triangular shape and are at the bottom of the figure.

The results gathered from the multi-criteria benchmark have been summarized in Table II. First, we define the solution quality difference metric as

$$\sigma_i = \frac{f_i^{\text{cplex}} - f_i^{\text{cp}}}{f_i^{\text{cplex}}}, \quad i \in 1, 2, 3 \tag{9}$$

[1] https://github.com/mdh-planner/Application-Offloading-in-E2C-Continuum

TABLE II

BENCHMARK RESULTS FOR MULTI-CRITERIA OPTIMIZATION, CPLEX VS. CP.

Instance	CP			Time [s]	CPLEX			Time [s]	σ_i		
	f_1	f_2	f_3		f_1	f_2	f_3		f_1	f_2	f_3
1	2	10834	6	0.5	2	10834	6	0.2	0	0	0
2	1	22872	9	1.3	1	22872	9	0.3	0	0	0
3	3	31354	21	573	3	31354	21	1.8	0	0	0
4	4	49073	26	2288	4	48168	26	13.8	0	1.9 %	0
5	4	66334	30	2526	4	65276	30	18.1	0	1.6 %	0
6	7	68548	34	3090	7	64804	34	103	0	5.8 %	0
7	19	178847	49	81.5	13	89430	45	135	32%	50 %	8.2 %
8	42	189774	76	200	41	92926	75	2191	2.4 %	51 %	1.3 %
9	16	93837	59	3344	16	81658	59	315	0	13 %	0
10	26	257401	77	650	17	102553	69	1761	35 %	60 %	10.4 %

TABLE III

BENCHMARK RESULTS FOR SINGLE-CRITERIA OPTIMIZATION, CPLEX VS. CP.

Instance	**CPLEX**	**Time**	**CP**	**Time**	**CPLEX**	**Time**	**CP**	**Time**	**CPLEX**	**Time**	**CP**	**Time**
	f_1	**[s]**	f_1	**[s]**	f_2	**[s]**	f_2	**[s]**	f_3	**[s]**	f_3	**[s]**
1	2	0.04	2	0.05	9610	0.06	9610	0.5	5	0.1	5	0.2
2	1	0.1	1	0.06	17138	0.11	17138	72.2	3	0.1	3	0.4
3	3	0.5	3	0.6	21870	0.37	21870	464	13	0.6	13	1.5
4	4	1	4	2.8	30136	0.8	30371	1868	11	0.7	11	3.6
5	4	4.6	4	4.6	44808	11.3	46534	764	17	2.3	17	43.8
6	7	46.9	7	66.2	53688	22.7	55159	3123	27	15.4	27	12.1
7	13	41.8	13	29.6	77572	154	82144	3600	37	51.9	37	613
8	41	217	41	64.9	90247	3600	99204	3552	73	88.7	73	60.1
9	16	87.4	16	94	69492	3600	75486	3231	45	62.9	45	233
10	17	230	17	1891	76090	34.5	127002	837	48	201	49	2282

where f_i^{cplex} represents the solution that is being compared from the CPLEX solver (e.g., f_1, f_2, or f_3), and f_i^{cp} represents the solution obtained from CP optimizer. For the smaller instances (1–3), both of the solvers were able to find the optimal solution. However, CPLEX can reach optimality in significantly less time for Instance 3. Looking at instances 4–6, it can be noticed that CP is being outperformed when it comes to objective f_2. The difference is in the range of 1.6–5.8% depending on the instance. The values of objectives f_2 and f_3 were optimized to optimality. However, the time solvers take differs by 2 orders of magnitude. At this point, it becomes obvious that CPLEX is a clear winner. This is even more exaggerated in instances 7–10, where CP cannot keep up with CPLEX in terms of f_1 and f_3, except Instance 9, where the difference in performance goes up to 60%.

Table III compares CPLEX and CP in the single objective optimization. We used the same instances from the previous benchmark, but we used a single criterion instead of multi-criteria optimization this time. The benchmark is done three times, once for each criterion. The results from CPLEX and CP, for the objective function f_1, where the goal is to minimize the total number of nodes used, are the same in terms of solution quality. Both solvers found an optimal solution in a comparable amount of time, except, for instance, 10, where CP took significantly more time than CPLEX to reach the optimal solution. However, in other instances, e.g., 2, 7, and 8, CP can get the optimal solution faster than CP. This has never been the case in the multi-criteria benchmark. When it comes to f_2, where the goal is to minimize the overall latency of the system, CPLEX outperforms CP by a large margin, both in terms of solution quality and convergence time, except in the first three test instances. Finally, in the case of optimization of f_3, it is similar to f_1. Both solvers can find the optimal solutions, except CP, in instance 10. However, in most cases, CPLEX performs better in terms of convergence time, except in instances 6 and 8.

From these two benchmarks, it can be concluded that when it comes to optimizing latency in this problem, the CPLEX is a clear winner. On the other hand, when it comes to maximizing the number of green nodes used or the total number of used nodes, both CP and CPLEX can be a reasonable choice, even though CPLEX does converge faster on average. It is also interesting that even though in multi-criteria optimization, CP performed worse than CPLEX, in single-criterion optimization, CP can still come ahead in some instances. Both approaches proved usable for this problem on a reasonably large scale since the largest instance had 600 applications to be offloaded to 120 nodes.

In addition, it can be observed that even though in our test instances the number of green nodes is between 27 and 60 % (Table I), the results in Table II, for CPLEX solver, show that the usage of green nodes, calculated as $(f_3 - f_1)/f_3$, in the solutions, is maximized and is in the range of 67 - 89 %, except for instance 8, where the equipment required by the applications is mostly provided by fossil-fueled nodes, in which case only 45% of the used nodes are green. Averaged over all 10 instances, CPLEX solutions include around 76% of green nodes, while CP has a lower average of around

73.7%. Depending on the nodes' consumption, the difference of 2.3% can greatly impact reducing the CO_2 footprint by having a better-optimized application offloading.

V. Conclusion

Application offloading is an important problem in the Cloud-based domain. It allows end devices to offload some or all of the computational load to an Edge-to-Cloud continuum while maintaining the required latency. However, for the offloading to be effective, it is necessary to have a model of the problem and good solvers that can solve a given problem in a reasonable time. To meet both demands from users and service providers, it is necessary to optimize more than one criterion. Also, some regulations might need to be followed, e.g., the allowed CO_2 footprint per node.

To address the issues above, we have proposed a mathematical model that describes the problem. In addition, we have implemented said model in two commercially available solvers, CPLEX and CP Optimizer, to understand if they might be applicable in this scenario. We also compared them to understand their benefits and shortcomings and determine which is more suitable for this specific application. We performed a benchmark on the series of randomly generated test instances, which shows the superiority of CPLEX in minimizing the overall latency of the system and in multi-criteria optimization. However, both CP and CPLEX solvers were effective in optimizing the total number of nodes and the number of fossil-fueled nodes used in the E2C continuum in single-objective optimization.

References

[1] K. Akherfi, M. Gerndt, and H. Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied computing and informatics*, 14(1):1–16, 2018.

[2] Ericsson. Ericsson mobility report (november 2022), 2022. Accessed: 2023-03-30.

[3] S. Forti and A. Brogi. Green application placement in the cloud-iot continuum. In *Practical Aspects of Declarative Languages: 24th International Symposium, PADL 2022, Philadelphia, PA, USA, January 17–18, 2022, Proceedings*, pages 208–217. Springer, 2022.

[4] H. Isermann. Linear lexicographic optimization. *Operations-Research-Spektrum*, 4(4):223–228, 1982.

[5] V. Jain and B. Kumar. Optimal task offloading and resource allotment towards fog-cloud architecture. In *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 233–238. IEEE, 2021.

[6] M. Jansen, A. Al-Dulaimy, A. V. Papadopoulos, A. Trivedi, and A. Iosup. The SPEC-RG reference architecture for the edge continuum. In *23rd IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2023.

[7] B. Johansson, M. Rågberger, A. V. Papadopoulos, and T. Nolte. Kubernetes orchestration of high availability distributed control systems. In *23rd IEEE International Conference on Industrial Technology (ICIT)*, pages 1–8, Aug. 2022.

[8] B. Kar, W. Yahya, Y.-D. Lin, and A. Ali. Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: A survey. *IEEE Communications Surveys & Tutorials*, 2023.

[9] P. W. Khan, K. Abbas, H. Shaiba, A. Muthanna, A. Abuarqoub, and M. Khayyat. Energy efficient computation offloading mechanism in multi-server mobile edge computing—an integer linear optimization approach. *Electronics*, 9(6):1010, 2020.

[10] L. Lin, X. Liao, H. Jin, and P. Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

[11] Y.-D. Lin, J.-C. Hu, B. Kar, and L.-H. Yen. Cost minimization with offloading to vehicles in two-tier federated edge and vehicular-fog systems. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–6. IEEE, 2019.

[12] Linux Foundation. State of the edge 2021, 2021. Accessed: 2023-03-30.

[13] J. Liu, S. Guo, Q. Wang, C. Pan, and L. Yang. Optimal multi-user offloading with resources allocation in mobile edge cloud computing. *Computer Networks*, 221:109522, 2023.

[14] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials*, 19(3):1628–1656, 2017.

[15] A. Mahjoubi, K.-J. Grinnemo, and J. Taheri. Ehga: A genetic algorithm based approach for scheduling tasks on distributed edge-cloud infrastructures. In *2022 13th International Conference on Network of the Future (NoF)*, pages 1–5. IEEE, 2022.

[16] M. Maray and J. Shuja. Computation offloading in mobile cloud computing and mobile edge computing: survey, taxonomy, and open issues. *Mobile Information Systems*, 2022, 2022.

[17] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE communications surveys & tutorials*, 20(1):416–464, 2017.

[18] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam. Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. In *Proceedings of the first international workshop on Mobile cloud computing & networking*, pages 19–26, 2013.

[19] P. Patros, J. Spillner, A. V. Papadopoulos, B. Varghese, O. Rana, and S. Dustdar. Toward sustainable serverless computing. *IEEE Internet Computing*, 25(6):42–50, 2021.

[20] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, and B.-S. Kim. Resource scheduling in edge computing: Architecture, taxonomy, open issues and future research directions. *IEEE Access*, 2023.

[21] Y. Ren, Y. Sun, and M. Peng. Deep reinforcement learning based computation offloading in fog enabled industrial internet of things. *IEEE Transactions on Industrial Informatics*, 17(7):4978–4987, 2020.

[22] F. A. Salaht, F. Desprez, and A. Lebre. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, 53(3):1–35, 2020.

[23] F. A. Salaht, F. Desprez, A. Lebre, C. Prud'Homme, and M. Abderrahim. Service placement in fog computing using constraint programming. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 19–27. IEEE, 2019.

[24] P. Shaw. A constraint for bin packing. In M. Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 648–662, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[25] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[26] P. Skarin, J. Eker, and K.-E. Årzén. A cloud-enabled rate-switching mpc architecture. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3151–3158, 2020.

[27] P. Skarin, J. Eker, M. Kihl, and K.-E. Årzén. Cloud-assisted model predictive control. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 110–112, 2019.

[28] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl. Control-over-the-cloud: A performance study for cloud-native, critical control systems. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pages 57–66, 2020.

[29] S. Smolka and Z. Á. Mann. Evaluation of fog application placement algorithms: a survey. *Computing*, 104(6):1397–1423, 2022.

[30] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor. Handling service allocation in combined fog-cloud scenarios. In *2016 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2016.

[31] W. Tärneberg, E. Fitzgerald, M. Bhuyan, P. Townend, K.-E. Årzén, P.-O. Östberg, E. Elmroth, J. Eker, F. Tufvesson, and M. Kihl. The 6g computing continuum (6gcc): Meeting the 6g computing challenges. In *1st International Conference on 6G Networking*, pages 1–5, 2022.

[32] W. Tärneberg, A. V. Papadopoulos, A. Mehta, J. Tordsson, and M. Kihl. Distributed approach to the holistic resource management of a mobile cloud network. In *1st International Conference on Fog and Edge Computing (ICFEC)*, pages 51–60, May 2017.

[33] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919, 2017.