Energy-Aware Capacity Provisioning and Resource Allocation in Edge Computing Systems

Tayebeh Bahreini[1], Hossein Badri[2], and Daniel Grosu[3]

[1] Department of Computer Science
Wayne State University
tayebeh.bahreini@wayne.edu
[2] Department of Computer Science
Wayne State University
hossein.badri@wayne.edu
[3] Department of Computer Science
Wayne State University
dgrosu@wayne.edu

Abstract. Energy consumption plays a key role in determining the cost of services in edge computing systems and has a significant environmental impact. Therefore, minimizing the energy consumption in such systems is of critical importance. In this paper, we address the problem of energy-aware optimization of capacity provisioning and resource allocation in edge computing systems. The main goal is to provision and allocate resources such that the net profit of the service provider is maximized, where the profit is the difference between the aggregated users' payments and the total operating cost due to energy consumption. We formulate the problem as a mixed integer linear program and prove that the problem is NP-hard. We develop a heuristic algorithm to solve the problem efficiently. We evaluate the performance of the proposed algorithm by conducting an extensive experimental analysis on problem instances of various sizes. The results show that the proposed algorithm has a very low execution time and is scalable with respect to the number of users in the system.

1 Introduction

Efficient utilization of computing resources has always been an important challenge for service providers, leading to significant efforts on developing solutions, either in the form of new technology or new ways to enhance the efficiency of existing technologies. Edge Computing (EC) is the latest technology developed to mitigate some of the existing challenges in cloud computing. In fact, the high latency in cloud computing systems which stems from the long distance between cloud servers and the end user, triggered the idea of EC systems, that is, bringing computing resources closer to the end user. EC systems are expected to improve the Quality of Service (QoS) by bringing servers closer to the end user, but when it comes to the cost of services, these systems face an important challenge. The operating cost of EC systems is higher than that of the remote clouds, due to

the small servers which are distributed across the network. In addition, in EC systems, a larger number of providers compete to provide services at a lower cost, and as a result, obtain a higher market share. It might not be quite easy to lower the investment costs, but when it comes to the operating costs, optimizing the energy consumption would be a promising way to reduce them. Studies show that about 25% of the operating costs of cloud data centers is attributed to energy consumption [10].

Given these facts, researchers have approached the resource provisioning problem in distributed systems from different perspectives. A variety of algorithms have been proposed to efficiently allocate users' requests to the cloud servers with an emphasis on reducing energy consumption of data centers [4, 6, 7, 18]. Several researchers considered task/workload consolidation as a strategy for reducing the energy consumption [12, 15]. Minimizing the total number of active servers is another strategy considered by some researchers. Torres et al. [16] proposed a technique to minimize the total number of active servers without degradation of QoS. Beloglazov et al. [5] and Hameed et al. [11] survey the research on energy-efficient cloud computing systems.

Several studies have focused on computation offloading in EC systems. Trinh et al. [17] studied the impact of computation offloading on energy consumption in EC systems. Chen et al. [8] developed a game theoretic approach for computation offloading in a multi-channel wireless network to minimize the energy consumption of mobile devices and the processing time of applications. Sardellitti et al. [14] and Zhang et al. [19] developed algorithms for decision making on the computational resources and the radio resources to minimize the system energy cost while meeting latency constraints. Bahreini and Grosu [3] designed an iterative matching algorithm for efficient placement of multi-component applications in edge computing systems. These approaches have only focused on resource allocation and did not investigate the capacity provisioning in EC systems.

Anglano et al. [2] developed an algorithm for resource allocation and capacity provisioning in EC systems with the aim of maximizing the profit of the system. To the best of our knowledge, this research is the first work addressing the integrated capacity provisioning and resource allocation in EC systems that takes the energy consumption into account. However, their proposed algorithm is based on solving a mixed-integer linear program which might not be feasible to solve within a reasonable amount of time for large size problems.

Our contributions. In this paper, we address the capacity provisioning and resource allocation problem in EC systems with the aim of maximizing the net profit of the provider while taking into account the energy consumption of the system. Our main contributions are as follows: (i) develop an energy-aware integrated formulation of the capacity provisioning and resource allocation problem for edge computing systems; (ii) prove that the energy-aware provisioning and resource allocation problem in edge computing systems is NP-hard; (iii) design an efficient heuristic algorithm to solve the problem; and (iv) perform an exten-

sive experimental analysis that shows that the proposed algorithm is scalable with the number of users and produces solutions that are close to optimal.
Organization. The rest of the paper is organized as follows. In Section 2, we define the problem and characterize its complexity. In Section 3, we describe our proposed heuristic algorithm. In Section 4, we define the experimental setup and discuss the experimental results. In Section 5, we conclude the paper and propose possible directions for future work.

2 Energy-aware capacity provisioning and resource allocation problem

In this section, we formulate the *Energy-aware Capacity Provisioning and Resource Allocation* (ECPRA) problem in EC systems. We consider an EC system owned and managed by a single provider that aims at maximizing its net profit (i.e., the profit per unit of time). In this system, users' devices generate a high amount of data that needs real-time processing. To guarantee the QoS for requests, the provider deploys a set of powerful computing resources at the edge of the network. However, these resources are limited and the provider is not able to allocate all requests to the edge side. Therefore, some of the requests will be allocated to the cloud side. On the other hand, the operating cost of edge resources is relatively higher than the operating cost of the cloud resources which results in a higher price per unit of resource at the edge. The provider's goal is to allocate resources to users in order to maximize its net profit, which is the total payment of users minus the total operating cost of resources per unit of time.

We denote the edge/cloud levels by ℓ (i.e., $\ell = 1$ for the edge level, and $\ell = 2$ for the cloud level). The system is composed of M^ℓ physical nodes at each level. Users can request D types of resources. For the sake of making the presentation simpler, we assume that $D = 3$, that is, there are three types of resources that a user can request: CPU (cores) ($k = 1$), memory ($k = 2$), and storage ($k = 3$). The capacity of node h at level ℓ for the resource of type k is denoted by C^ℓ_{hk}. We consider N users requesting resources as containers from the provider. The request of user i consists of Q_i containers and is denoted by $R_i = \{r_{i1k}, \ldots, r_{iQ_ik}\}$, where r_{ijk} is the amount of resource of type k requested by user i for container j. As an example, suppose that user i's request is $R_i = \{\{4, 6, 0\}, \{2, 1, 5\}\}$. This means that the number of containers, Q_i, requested by user i, is two ($Q_i = 2$). The first container requires four cores, 6 GB of memory, and no storage, while the second container requires two cores, 1 GB of memory, and 5 GB of storage. The provider allocates a given container to a single node. Also, to have a consistent response time from the physical nodes, the whole request from a user is allocated at either edge or cloud level, but not at both.

Upon receiving the request, the provider decides how to provision resources and allocate the users' requests in order to maximize the total profit, where the profit is the difference between the payments received form the users and the operating cost. We consider that *the operating cost of a node is proportional to*

the energy consumption of that node which can be estimated by a linear function of CPU, memory, and disk utilization [13, 20]. Therefore, the energy consumption is captured in the objective function through the operating cost. The *operating cost* (due to energy consumption) of a powered-on node h at level ℓ is given by,

$$E_h^\ell = \delta_h^\ell + \sum_{k=1}^{3} u_{hk}^\ell \cdot \rho_{hk}^\ell \tag{1}$$

where, δ_h^ℓ is the operating cost of node h at level ℓ when it is idle, ρ_{hk}^ℓ is the operating cost of node h for the resource of type k when the resource is fully utilized, and u_{hk}^ℓ is the utilization rate of node h's resource of type k. The utilization u_{hk}^ℓ is given by,

$$u_{hk}^\ell = \frac{1}{C_{hk}^\ell} \sum_{i=1}^{N} \sum_{j=1}^{Q_i} z_{hij}^\ell \cdot r_{ijk} \tag{2}$$

where z_{hij}^ℓ is a binary variable associated with the allocation of container j of user i to node h at level ℓ. The value of this variable is 1, if container j of user i is allocated to node h at level ℓ; and 0, otherwise. Therefore, the *total operating cost* of the system is,

$$E = \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} x_h^\ell \cdot \delta_h^\ell + \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} \sum_{k=1}^{3} u_{hk}^\ell \cdot \rho_{hk}^\ell \tag{3}$$

where, x_h^ℓ is a binary decision variable associated with the status of node h at level ℓ. Variable x_h^ℓ is 1 if the node is powered on; and 0, otherwise. These decision variables determine how many servers will be turned on by the provider, and therefore represent the capacity provisioning decision.

The provider charges each user a certain amount of money per each unit of time. The amount of money depends on the level that the user's request is allocated and the amount of resources that user requested. Denoting the *unit price* of a resource of type k at level ℓ by π_k^ℓ, the *payment of user* i is defined as,

$$p_i = \sum_{j=1}^{Q_i} \sum_{k=1}^{3} \sum_{\ell=1}^{2} y_i^\ell \cdot \pi_k^\ell \cdot r_{ijk} \tag{4}$$

where, y_i^ℓ is a binary variable, $y_i^\ell = 1$ if user i is allocated at level ℓ; and 0, otherwise. Therefore, we define the *net profit*, Π, of the provider as the aggregated users payments minus the total operating cost of nodes,

$$\begin{aligned} \Pi = \sum_{i=1}^{N} \sum_{j=1}^{Q_i} \sum_{k=1}^{3} \sum_{\ell=1}^{2} y_i^\ell \cdot \pi_k^\ell \cdot r_{ijk} - \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} x_h^\ell \cdot \delta_h^\ell \\ - \sum_{i=1}^{N} \sum_{j=1}^{Q_i} \sum_{k=1}^{3} \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} z_{hij}^l \cdot \frac{r_{ijk} \cdot \rho_{hk}^\ell}{C_{hk}^\ell} \end{aligned} \tag{5}$$

Table 1: Notation

Notation	Description
N	Number of users.
M^ℓ	Number of physical nodes at level ℓ.
D	Number of resource types.
Q_i	Number of containers requested by user i.
r_{ijk}	Amount of resource of type k from container j of user i.
C^ℓ_{hk}	Capacity of node h at level l for resource of type k.
δ^ℓ_h	Operating cost of node h at level l in idle mode.
ρ^ℓ_{hk}	Operating cost of node h at level l for resource of type k fully utilized.
π^ℓ_k	Unit price of resource of type k at level ℓ.
x^ℓ_h	Binary decision variable; status of node h at level ℓ.
z^ℓ_{hij}	Binary decision variable; allocation of container j of user i.
y^ℓ_i	Binary decision variable; allocation of user i at level ℓ.

To simplify the equations for profit, we define the following parameters:

$$\omega^\ell_{hij} = \sum_{k=1}^{3} \frac{r_{ijk} \cdot \rho^\ell_{hk}}{C^\ell_{hk}} \qquad \text{and} \qquad \eta^\ell_i = \sum_{j=1}^{Q_i} \sum_{k=1}^{3} \pi^\ell_k \cdot r_{ijk} \tag{6}$$

Now, we formulate the *Edge Capacity Provisioning and Resource Allocation* (ECPRA) problem. Table 1 summarizes the notation that we use in our formulation. The mixed-integer linear program (MILP) formulation of ECPRA is as follows,

ECPRA-MILP:

$$\text{Maximize} \quad \sum_{i=1}^{N} \sum_{\ell=1}^{2} y^\ell_i \cdot \eta^\ell_i - \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} x^\ell_h \cdot \delta^\ell_h - \sum_{i=1}^{N} \sum_{j=1}^{Q_i} \sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} z^\ell_{hij} \cdot \omega^\ell_{hij} \tag{7}$$

subject to:

$$\sum_{i=1}^{N} \sum_{j=1}^{Q_i} z^\ell_{hij} \cdot r_{ijk} \leq x^\ell_h \cdot C^\ell_{hk} \quad \forall h, \forall k, \forall \ell \tag{8}$$

$$y^\ell_i \cdot Q_i \leq \sum_{h=1}^{M^\ell} \sum_{j=1}^{Q_i} z^\ell_{hij} \quad \forall i, \forall \ell \tag{9}$$

$$\sum_{\ell=1}^{2} \sum_{h=1}^{M^\ell} z^\ell_{hij} \leq 1 \quad \forall i, \forall j \tag{10}$$

$$x^\ell_h \in \{0,1\}, \quad y^\ell_i \in \{0,1\} \quad \forall h, \forall \ell \tag{11}$$

$$z^\ell_{hij} \in \{0,1\} \quad \forall i, \forall j, \forall \ell, \forall h \tag{12}$$

Equation (7) is the objective function which is the total net profit of the provider. Constraints (8) ensure that the total allocated resources of each type

on node h at level ℓ does not exceed the available capacity of that type of resource. Note that these constraints also determine the mode of the node; if $x_h^\ell = 1$, the corresponding node is on; otherwise, the node is off and no request can be allocated to it. Constraints (9) guarantee that user i is allocated at level ℓ if and only if the whole request of this user is allocated to the nodes situated at level ℓ. Constraints (10) ensure that no container is allocated to more than one node. Finally, constraints (11) - (12) guarantee the integrality of the decision variables.

2.1 Complexity of ECPRA

We prove that ECPRA is an NP-hard problem, that is, it is not solvable in polynomial time, unless $P = NP$. We prove this claim by showing that a special case of this problem is NP-hard.

Theorem. The ECPRA problem is NP-hard.

Proof. Let us consider a special case of ECPRA in which there is only one user in the system, and there exists only one level of resources. We call this problem ECPRA-S. We show that ECPRA-S is an NP-hard problem. Then, we conclude that ECPRA, which is the general case of ECPRA-S, is NP-hard as well.

From Equation (7), the objective of ECPRA-S for user i at level ℓ is,

$$\text{Maximize} \quad y_i^\ell \cdot \eta_i^\ell - \sum_{h=1}^{M^\ell} x_h^\ell \cdot \delta_h^\ell - \sum_{j=1}^{Q_i} \sum_{h=1}^{M^\ell} z_{hij}^\ell \cdot \omega_{hij}^\ell \tag{13}$$

Since there is only one user in the system, the first term in the objective function (η_i^ℓ) has a fixed value, that is, it does not have any effects on the solution. Therefore, we can ignore it. Furthermore, for our purpose, we convert the objective from maximization to minimization. Since i and ℓ have a fixed value, for the sake of readability, we define binary variables $\bar{x}_h$ and $\bar{y}_{hj}$, where $\bar{x}_h = x_h^\ell$ and $\bar{z}_{hj} = z_{hij}^\ell$. We also define parameters, $\bar{\delta}_h = \delta_h^\ell$, $\bar{\omega}_{hj} = \omega_{hij}^\ell$, $\bar{r}_{jk} = r_{ijk}$ and $\bar{C}_{hk} = C_{hk}^\ell$. Thus, we can formulate ECPRA-S as,

$$\text{Minimize} \quad \sum_{h=1}^{M^\ell} \bar{x}_h \cdot \bar{\delta}_h + \sum_{j=1}^{Q_i} \sum_{h=1}^{M^\ell} \bar{z}_{hj} \cdot \bar{\omega}_{hj} \tag{14}$$

subject to:

$$\sum_{j=1}^{Q_i} \bar{z}_{hj} \cdot \bar{r}_{jk} \leq \bar{x}_h \cdot \bar{C}_{hk} \quad \forall h, \forall k \tag{15}$$

$$\sum_{h=1}^{M^\ell} \bar{z}_{hj} \leq 1 \quad \forall j \tag{16}$$

$$\bar{x}_h \in \{0, 1\} \quad \forall h \tag{17}$$

$$\bar{z}_{hj} \in \{0, 1\} \quad \forall h, \forall j \tag{18}$$

We observe that ECPRA-S is the general case of the Capacitated Facility Location (CFL) problem [9], where instead of having a single type of goods, each facility provides different types of goods. In fact, in CFL, there is a set of facilities (nodes), each facility provides a single type of goods (resources) with a limited capacity (Constraints (15)). There is also a set of clients (set of containers), and a client j has a demand, $\bar{r}_{jk}$. The whole demand of a client must be assigned to a single facility (Constraints (16)). Each facility has a fixed cost to be opened, $\bar{\delta}_h$. Satisfying the demand of each client from each facility has a different cost, $\bar{\omega}_{hj}$. The goal is to select a subset of facilities to open, in order to minimize the sum of the cost of the assignment, plus the sum of facilities' opening cost (Equation (14)). CFL is a well-known NP-hard problem [9]. Since ECPRA-S is a generalization of CFL, ECPRA-S is NP-hard as well. Furthermore, ECPRA is a generalization of ECPRA-S. Therefore, ECPRA is an NP-hard problem.

3 A greedy algorithm for ECPRA

ECPRA is NP-hard and therefore, it is not solvable in polynomial time, unless $P = NP$. We give up on optimality and develop a greedy algorithm, called G-ECPRA, that provides a suboptimal solution to ECPRA in polynomial time. Our greedy algorithm is an iterative algorithm; and in each iteration, the allocation of only one user is determined. In fact, in each iteration of the algorithm, a user that maximizes the revenue of the system is selected, and then, the algorithm finds an allocation for that user that minimizes the operating cost of the system.

The proposed algorithm for solving the ECPRA problem is given in Algorithm 1. The algorithm has as input the vector of users' requests and the capacity of the nodes at each level, and determines the allocation of these requests. The output of the algorithm consists of the profit of the provider, $\mathit{\Pi}$, and the allocation matrices $X = \{x_h^\ell\}$, $Y = \{y_i^\ell\}$, and $Z = \{z_{hij}^\ell\}$.

First, G-ECPRA initializes the allocation matrices X, Y, and Z, and the status matrix $S = \{s_h^\ell\}$ (Line 1). The status matrix indicates the status of the nodes after the last iteration of the algorithm, that is, $s_h^\ell = 0$ if node h at level ℓ is turned on, and $s_h^\ell = 1$ if that node is off. Initially, this matrix is set to 1, that is, no node is selected to be turned on. G-ECPRA then determines the average revenue, $\mathit{\Gamma}_i$, that each user can bring to the system (Lines 2-3). It sorts users in non-increasing order of $\mathit{\Gamma}_i$ (Line 4). Then, in each iteration of the algorithm, an unallocated user with the maximum $\mathit{\Gamma}_i$ is chosen in order to maximize the revenue of the provider. In this step, Algorithm G-CFL is called twice to determine the possible allocations for the current user at both the edge level and the cloud level (Lines 6-7). G-CFL gets the request of the user, the current capacity at level ℓ, and the status of nodes, S, and finds an allocation for the user at level ℓ such that the operating cost is minimized. In fact, G-CFL tries to find a solution for ECPRA-S (we will explain this algorithm in more details later). $\bar{X}^\ell$ and $\bar{Z}^\ell$ are temporary matrices corresponding to the output matrices $\bar{X}$ and $\bar{Z}$ obtained by G-CFL for the current user at level ℓ, and $cost^\ell$ is the cost of allocating the current user at level ℓ.

Algorithm 1 G-ECPRA

Input: Users' requests: $\{R_1, \ldots, R_N\}$
Nodes' capacity: $C = \{C^{\ell}_{hk}\}$

1: $\Pi \leftarrow 0$, $X \leftarrow 0$, $Y \leftarrow 0$, $Z \leftarrow 0$, $S \leftarrow 1$
2: **for** $i = 1 \ldots N$ **do**
3: $\quad \Gamma_i \leftarrow \sum_{j=1}^{Q_i} \sum_{k=1}^{3} \frac{(\pi^1_k + \pi^2_k)}{2} \cdot r_{ijk}$
4: sort users in non-increasing order of Γ_i
5: **for** $i = 1 \ldots N$ **do**
6: $\quad$ **for** $\ell = 1 \ldots 2$ **do**
7: $\quad\quad \{\bar{X}^{\ell}, \bar{Z}^{\ell}, cost^{\ell}\} \leftarrow$ G-CFL$(R_i, C^{\ell}, S^{\ell})$
8: $\quad\quad \Pi^{\ell} \leftarrow \eta^{\ell}_i - cost^{\ell}$
9: $\quad \ell^* \leftarrow \text{argmax}_{\ell \in \{1,2\}} \Pi^{\ell}$
10: $\quad$ **if** $\Pi^{\ell^*} > 0$ **then**
11: $\quad\quad \Pi \leftarrow \Pi + \Pi^{\ell^*}$
12: $\quad\quad y^{\ell^*}_i \leftarrow 1$
13: $\quad\quad$ **for** $h \in M^{\ell^*}$ **do**
14: $\quad\quad\quad x^{\ell^*}_h \leftarrow \bar{x}^{\ell^*}_h$
15: $\quad\quad\quad$ **for** $j = 1 \ldots Q_i$ **do**
16: $\quad\quad\quad\quad z^{\ell^*}_{hij} \leftarrow \bar{z}^{\ell^*}_{hj}$

Output: X, Y, Z, Π

G-ECPRA determines the possible contribution to the profit by the current user, $\Pi^{\ell} = \eta^{\ell}_i - cost^{\ell}$ (Line 8). Then, the algorithm picks the level that yields the maximum profit (Line 9). If the profit at this level is positive, it means that G-CFL has found a feasible allocation for this user. In this case, the allocation matrices X, Y, Z, and the profit of the system are updated (Lines 10-16). If the profit is negative, it means that G-CFL has not found a feasible allocation for the user. Therefore, the allocation matrices will not be updated. This procedure is repeated until all users are considered.

The G-CFL algorithm, presented in Algorithm 2, finds an allocation for user i at level ℓ with the minimum operating cost. In fact, G-CFL solves the ECPRA-S problem. However, in the problem that G-CFL solves, some nodes might have been turned on due to the allocation of the previous users. Therefore, if any container of the current user is allocated on these nodes, no fixed cost δ^{ℓ}_h, will be added to the system.

G-CFL has as inputs the request of user i, the current capacity of the nodes at level ℓ, and the status matrix. It determines the allocation for user i. The output of G-CFL is the cost of allocating user i at level ℓ, and the allocation matrices $\bar{Z} = \{\bar{z}_{jk}\}$ and $\bar{X} = \{\bar{x}_h\}$.

First, G-CFL creates a set of unallocated containers, $\mathcal{R}$. Initially, $\mathcal{R}$ contains all user i's containers (Line 2). Then, the algorithm computes the cost of assigning each container on each node. For this purpose, function available is called (Line 6) to check whether node h has enough capacity for container j. If there are enough resources, then the cost of the assignment is given by $s^{\ell}_h \cdot \delta^{\ell}_h + \omega^{\ell}_{hij}$. Oth-

Algorithm 2 G-CFL

Input: Request of user i; $R_i = \{r_{ijk}\}$
Nodes' capacities at level ℓ; $C^\ell = \{C^\ell_{kh}\}$
Status of nodes at level ℓ; $S^\ell = \{s^\ell_h\}$

1: $\Pi \leftarrow 0$, $\bar{X} \leftarrow 0$, $\bar{Z} \leftarrow 0$
2: $\mathcal{R} \leftarrow \{1, \ldots, Q_i\}$
3: $\bar{C} \leftarrow C^\ell$
4: **for** each $j \in \mathcal{R}$ **do**
5: **for** $h = 1, \ldots, M^\ell$ **do**
6: **if** available$(\widehat{C}_h, j)$ **then**
7: $cost_{hj} \leftarrow \bar{x}^\ell_h \cdot \delta^\ell_h + \omega'_{hj}$
8: **else**
9: $cost_{hj} \leftarrow \infty$
10: **while** $\mathcal{R} \neq \emptyset$ **do**
11: $(h^*, j^*) \leftarrow \text{argmin}_{(h,j)}(cost_{hj})$
12: **if** $cost_{h^*j^*} \neq \infty$ **then**
13: $\bar{C}_{h^*} \leftarrow \bar{C}_{h^*} - r_{ij}$
14: $\bar{z}_{h^*j^*} \leftarrow 1$
15: $\bar{x}_{h^*} \leftarrow 1$
16: $cost \leftarrow cost + cost_{h^*j^*}$
17: remove j^* from $\mathcal{R}$
18: **for** each $j \in \mathcal{R}$ **do**
19: **if** available$(\bar{C}_{h^*}, j)$ **then**
20: $cost_{h^*j} \leftarrow \omega'_{h^*j}$
21: **else**
22: $cost_{h^*j} \leftarrow \infty$
23: **else**
24: $cost \leftarrow \infty$
25: **break**;
26: **if** $cost \neq \infty$ **then**
27: **for** $h = 1, \ldots, M^\ell$ **do**
28: $C^\ell_h \leftarrow \bar{C}_h$
29: **if** $\bar{x}_h = 1$ **then**
30: $s^\ell_h \leftarrow 0$

Output: $\bar{X}$, $\bar{Z}$, $cost$

erwise, the cost of the assignment is infinity, which means that the assignment is infeasible (Lines 4-9).

Then, G-CFL assigns containers to the nodes iteratively. In each iteration, it chooses a pair of node and container (h^*, j^*) that has the minimum value of assignment cost (Line 11). If $cost_{h^*j^*}$ is not infinity, it means that assigning container j^* to node h^* at level ℓ^* is feasible. In this case, the algorithm updates the capacity of the node, temporarily. Matrices $\bar{X}$ and $\bar{Z}$ and the cost of the system are also updated (Lines 12-16). Then, the algorithm removes container j^* from $\mathcal{R}$ (Line 17). After that, it updates the cost of assigning each remaining container to node $S^\ell_{h^*}$. Since this node is on now, the fixed cost must not be

considered in any other assignments of this node. The algorithm also checks if by this assignment, there are not enough resources for a container, then the cost of the further assignment is set to infinity (Lines 19-22). If the cost of assignment for (h^*, j^*) is infinity, it means that the minimum assignment cost is infinity. Therefore, the algorithm could not find a feasible assignment for this container and any other containers. Thus, it sets the total assignment cost to infinity, exits from the loop, and does not continue finding allocations for other containers (Lines 23-26). Outside the loop, the algorithm checks the value of *cost*. If it is not infinity, the allocation matrix, status matrix, and capacity are updated (Lines 26-30).

We now analyze the time complexity of the algorithm. In the analysis we use the following notation: $M = \max_\ell M^\ell$ and $Q = \max_i Q_i$. In G-CFL, the initialization (Lines 1-3) takes $\mathcal{O}(DMQ)$. The while loop of G-CFL takes $\mathcal{O}((M+D)Q^2)$. Since D (the number of resource types) is small compared to M and Q, the time complexity of G-CFL is $\mathcal{O}(MQ^2)$. The most time consuming part of G-ECPRA is the call to G-CFL for each user and for each level. Therefore, the time complexity of G-ECPRA is $\mathcal{O}(NMQ^2)$.

4 Experimental results

In this section, we present an experimental analysis of the performance of our proposed algorithm, G-ECPRA. We compare the performance of G-ECPRA with the optimal solution obtained by solving ECPRA-MILP with CPLEX [1]. Then, we investigate the scalability of G-ECPRA for large size instances. In the following, we describe the experimental setup and analyze the experimental results.

4.1 Experimental setup

We generate several problem instances with different values for the number of users, N, the number of resource types, D, and the amount of resources requested by each user, r_{ijk}. In the first set of experiments, we compare the performance of G-ECPRA with that of the CPLEX solver. Since the CPLEX solver cannot solve large size problem instances, we perform this analysis on relatively small size problem instances. We assume that there are a few nodes available at the edge level and the cloud level ($M^1 = 5$, $M^2 = 10$). For these problem instances, the number of users varies from 10 to 100, and there are four types of resources, i.e., CPU, memory, storage, and bandwidth. The value of r_{ijk} is chosen from a uniform distribution $U[0, RB]$, where RB is an upper bound for r_{ijk}.

Since the objective of ECPRA-MILP is to maximize the profit, we generate instances according to a metric called *price to cost ratio* (PCR). This metric is defined as the ratio of the average price per unit of resource to the average cost per unit of resource:

$$PCR = \frac{\sum_{k=1}^{3}\sum_{\ell=1}^{2}\sum_{h=1}^{M^\ell} C_{kh}^{\ell} \cdot \pi_k^\ell}{\sum_{\ell=1}^{2}\sum_{h=1}^{M^\ell}(\delta_h^\ell + \sum_{k=1}^{3} C_{kh}^{\ell} \cdot \rho_{kh}^{\ell})} \tag{19}$$

Table 2: Simulation parameters

Param.	Distribution	Param.	Distribution
C^1_{hk}	$U[1, 300]$		$PCR \approx 1 : U[1, 6]$
C^2_{hk}	$U[30, 300]$	π^1_k	$PCR \approx 2 : U[3, 10]$
Q_i	$U[1, 5]$		$PCR \approx 7 : U[10, 35]$
r_{ijk}	$U[0, RB]$		$PCR \approx 20 : U[40, 120]$
δ^1_h	$U[5, 50]$		$PCR \approx 1 : U[1, 3]$
δ^2_h	$U[1, 40]$	π^2_k	$PCR \approx 2 : U[1, 5]$
ρ^1_{kh}	$U[1, 10]$		$PCR \approx 7 : U[3, 20]$
ρ^2_{kh}	$U[1, 5]$		$PCR \approx 20 : U[10, 80]$

Table 2 shows the probability distributions used to generate the parameters in our experiments. We denote by U$[x, y]$ the uniform distribution on the interval $[x, y]$. To generate problem instances with different values of PCR, the price per unit of each type of resources is drawn from different distributions.

The performance of G-ECPRA is evaluated by computing the profit ratio, Π^r, which is the ratio of the value of the solution obtained by G-ECPRA, denoted by Π, and that of the optimal solution obtained by CPLEX, denoted by Π^*, i.e., $\Pi^r = \frac{\Pi}{\Pi^*}$.

In the second set of experiments, we investigate the scalability of G-ECPRA for large size problem instances. We consider a system with 50 servers at the edge level, and 100 servers at the cloud level ($M^1 = 50$, $M^2 = 100$). There are four types of resources, and the number of users ranges from 100 to 1500.

The G-ECPRA algorithm is implemented in C++ and the experiments are conducted on an Intel 1.6GHz Core i5 system with 8 GB RAM. For solving G-ECPRA-MILP, we use the CPLEX 12 solver provided by IBM ILOG CPLEX optimization studio for academics initiative [1].

4.2 Experimental Analysis

We first investigate the effects of the number of users on the performance of G-ECPRA. For each value of the number of users, we generate two sets of instances with different values of PCR. In these problem instances, all parameters are identical except π^ℓ_k. The value of π^ℓ_k is chosen according to Table 2, such that in the first set, $PCR \approx 2$, and in the second set, $PCR \approx 20$. For these problem instances, $RB \approx 6$.

Figure 1a shows the execution time for each instance. We observe that for each number of users, the running time of CPLEX for an instance with $PCR \approx 20$ is less than the instance with $PCR \approx 2$. The reason behind this is that when the PCR is high, the effect of the energy cost of servers on the profit of the system is not very significant. Thus, the main problem is to decide only on how to place the requests of each user, either at the edge or at the cloud level, in order to maximize the total payments. Therefore, the CPLEX solver can solve

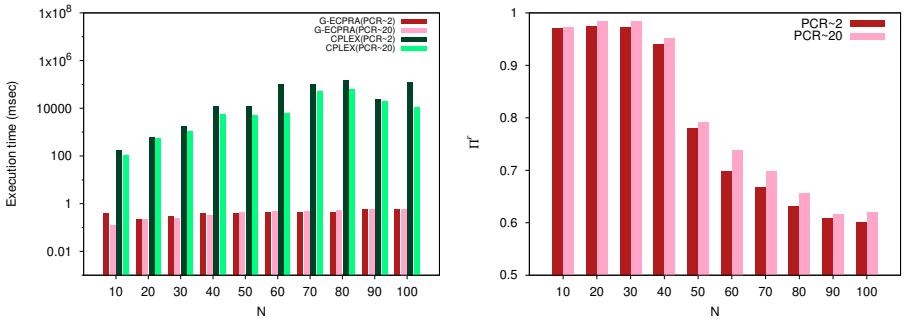
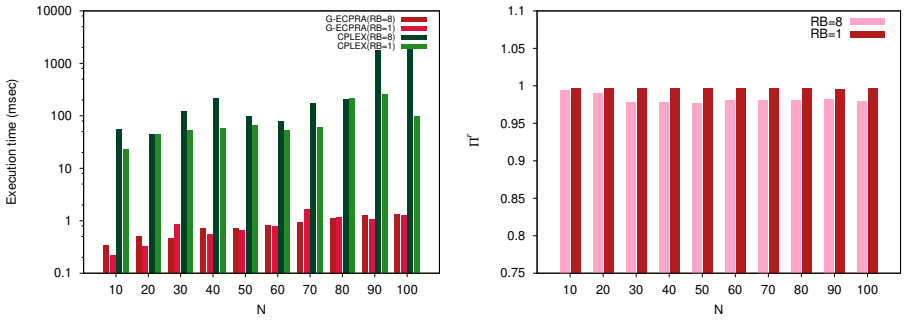


Fig. 1: The effect of PCR on the execution time and profit ratio (small-size instances)

the problem faster than the case in which we have a balance between cost and the price of each unit of resources.

Another observation from Figure 1a is with respect to the impact of the number of users on the running time. The running time of CPLEX (represented in the figure using a logarithmic scale) increases exponentially, while that of G-ECPRA increases linearly. Figure 1b shows the profit ratio for each of these instances. As we observe, the profit ratio for instances with $PCR \approx 20$ is higher than that for the instances with $PCR \approx 2$. The reason behind this observation is that with $PCR \approx 20$, minimizing the cost is not so important and the main focus is on maximizing the total payment. Therefore, G-ECPRA, which decomposes the main problem into two subproblems obtains solutions that are not far from solutions of the main problem. Another observation from Figure 1b is that for both instances, by increasing the number of users, the profit ratio decreases. Since the capacity of the nodes is identical for all instances, by increasing the number of users in the system, the allocation becomes more competitive and it becomes harder to decide how to allocate resources in order to maximize the profit. However, the solution obtained by G-ECPRA is good. For $N = 10$ the profit ratio is about 0.95 while for $N = 100$, it is about 0.6, which is still reasonable.

Next, we analyze the effects of the number of users on the performance of the algorithm. But this time, our analysis is based on two sets of problem instances with different values of the request bound, RB. In fact, we investigate the effects of RB on the quality of solutions. In the first set, each container has at most one unit of each type of resources ($RB = 1$), while in the second set, each container can contain up to eight units of each type of resources ($RB = 8$). For these instances, $PCR \approx 2$. Figure 2a shows the running time of CPLEX and G-ECPRA for each instance. We observe that the running time of G-ECPRA does not change dramatically and is less than 1 millisecond for all instances. But the



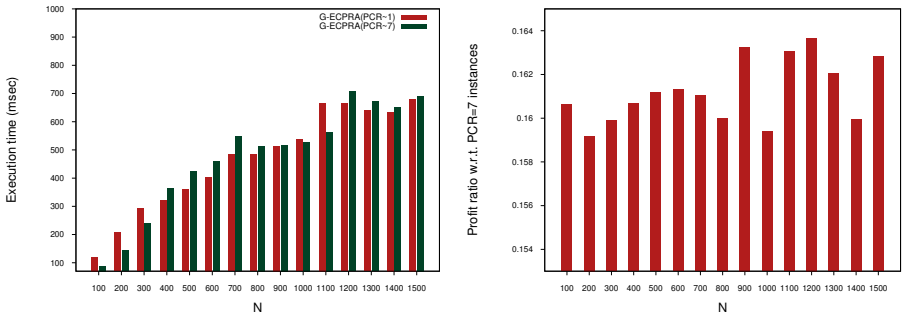
(a) (b)

Fig. 2: The effect of the request size bound, RB, on the execution time and profit ratio (small-size instances)

running time of CPLEX increases by increasing RB, and for some instances, it exceeds 2050 milliseconds. The reason behind this behavior is that the complexity of the problem increases when each container contains more than one unit of each type of resources. Figure 2b shows that the profit ratio decreases when RB increases. Since by increasing RB, the complexity of the problem increases, the solution obtained by G-ECPRA is farther from the optimal solution than in the case with smaller values of RB. However, the quality of solutions is still very good (the profit ratio is about 0.98).

In the next set of experiments, we investigate the performance of the algorithm for *large-scale problem instances*. We define two sets of instances with $PCR \approx 1$ and $PCR \approx 7$, respectively. Because for these sets of problem instances the size of problem is large, the CPLEX is unable to obtain the optimal solution in reasonable amount of time. Figure 3a shows the running time of our algorithm for these two instances. As we observe, by increasing the number of users, the running time of our algorithm also increases, but the increase is linear. Even for these large instances, the running time of our algorithm is under 800 milliseconds, thus, making it suitable for deployment in EC systems.

Since, the CPLEX solver is unable to solve these large size problem instances in a reasonable amount of time, we cannot compare the profit obtained by G-ECPRA with that of CPLEX. Figure 3b shows the profit ratio between the profit obtained for instances with $PCR \approx 1$ to the profit obtained for instances with $PCR \approx 7$. We observe that in all cases this ratio is about 0.16 which is very close to the ratio between the PCR of the two instance sets. This indicates that our algorithm has a stable behavior and that the value of PCR does not significantly affect the performance of the algorithm.



(a) (b)

Fig. 3: The effect of PCR on the execution time and profit ratio with respect to $PCR \approx 7$ instances (large-size instances)

5 Conclusion

There is an increasing concern about the energy consumption in edge computing systems, both from the perspective of the environmental impact as well as business competitiveness. In this paper, we proposed an energy-conscious approach for the capacity provisioning and resource allocation problem in edge computing systems. We proposed an MILP formulation of the problem and proved that the problem is NP-hard. In order to solve the problem efficiently, we proposed a heuristic algorithm and analyzed its performance. Our experimental analysis on different sizes and various configurations of the problem showed that the proposed greedy algorithm is competitive with the CPLEX solver in terms of both the computational time and the quality of solutions.

In future work, we plan to develop energy-aware auction based mechanisms for the capacity provisioning and resource planning in edge computing systems. Also, we plan to take the uncertainty of demands into account in the energy-aware capacity provisioning and resource allocation in edge computing systems.

References

1. IBM ILOG CPLEX V12.1 user's manual, 2009.
2. C. Anglano, M. Canonico, and M. Guazzone. Profit-aware resource management for edge computing systems. In *Proc. of the 1st International Workshop on Edge Systems, Analytics and Networking*, pages 25–30. ACM, 2018.
3. T. Bahreini and D. Grosu. Efficient placement of multi-component applications in edge computing systems. In *Proc. of the 2nd ACM/IEEE Symposium on Edge Computing*, pages 5:1–5:11, 2017.
4. A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

5. A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. In *Advances in Computers*, volume 82, pages 47–111. Elsevier, 2011.
6. R. Buyya, A. Beloglazov, and J. Abawajy. Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010.
7. J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review*, 35(5):103–116, 2001.
8. X. Chen, L. Jiao, W. Li, and X. Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, (5):2795–2808, 2016.
9. M. R. Garey and D. S. Johnson. *Computers and intractability, A guide to the theory of NP-Completeness*, volume 29. WH Freeman, New York, 2002.
10. A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.
11. A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, et al. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 98(7):751–774, 2016.
12. Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.
13. S. Rivoire, P. Ranganathan, and C. Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8(2):32–39, 2008.
14. S. Sardellitti, G. Scutari, and S. Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.
15. S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proc. of the 2008 Conf. on Power Aware Computing and Systems*, volume 10, pages 1–5. San Diego, California, 2008.
16. J. Torres, D. Carrera, K. Hogan, R. Gavaldà, V. Beltran, and N. Poggi. Reducing wasted resources to help achieve green data centers. In *Proc. IEEE Int. Symp. on Parallel and Distributed Processing*, pages 1–8. IEEE, 2008.
17. H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan. Energy-aware mobile edge computing for low-latency visual data processing. In *Proc. of the 5th IEEE Int. Conf. on Future Internet of Things and Cloud*, pages 128–133, 2017.
18. A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 243–264. Springer-Verlag New York, Inc., 2008.
19. K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang. Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907, 2016.
20. Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proc. of the 9th Int. Conf. on Autonomic Computing*, pages 145–154, 2012.