A Migration Path Toward Green Edge Gaming

Francesco Spinelli[♯†], Vincenzo Mancuso[♯]
[♯]IMDEA Networks Institute, Madrid, Spain,
[†]Universidad Carlos III de Madrid, Madrid, Spain
E-mails: francesco.spinelli@imdea.org, vincenzo.mancuso@imdea.org

Abstract*—5G and beyond 5G networks will allow novel use cases by placing constrained computing nodes at the edge, following the Multi-access Edge Computing (MEC) paradigm. Edge nodes could be partially powered by intermittent renewable energies, leading to the possibility of having time-varying computing capacities. In this scenario, we tackle the problem of how to support gaming at the edge of the cellular network. Moving cloud-based games to the edge could be a premium service for end-users, thanks to reduced latency and higher bandwidth. The goal of our paper is to design a scheme that maximizes the utility of a service/infrastructure provider in a MEC scenario, with time-varying MEC nodes capacities powered by intermittent renewable energies. We formulate a *multi*-dimensional integer linear programming problem, proving that it is NP Hard in the strong sense. We prove that our problem is sub-modular and propose an efficient heuristic, `GREENING`, which considers the allocation of gaming sessions and their migration. Through simulations, we show that our heuristic achieves performance close to what achievable by a solver, except with extremely lower complexity, and performs near-optimally, 20% better than state-of-the-art algorithms in terms of system utility. We also show that our scheme is compliant with currently adopted standards by ETSI and meant to support novel networking principles like network slicing.

***Index Terms*—MEC, Edge computing, Edge gaming, green energy, renewable energies, beyond 5G, 5G, Resources allocation and migration**

I. Introduction

Cloud gaming allows users to leverage on a cloud infrastructure to play games, requiring only an internet connection and a screen (e.g. a TV screen, laptop or mobile phone). Games are located and processed in a cloud server, which streams the content to the end-user screen. While several past attempts were unsuccessful (e.g. OnLive) mainly due to the lack of infrastructure, nowadays cloud gaming is having a second life. It is a growing market (reaching by 2023 a total revenue of \$ 8 billion[1]) and many tech companies (among the others, Google with Google Stadia, NVidia with Geforce Now and AWS with Amazon Luna) are launching cloud gaming services within their network infrastructure. At the same time, one of the main features of 5G and beyond 5G networks is to place computing capabilities closer to end-users, at the edge of the network. In this scenario, paradigms such as Multi-access Edge Computing (MEC) [1] arose, allowing novel use cases or *verticals* [2] MEC gives the opportunity to exploit cloud gaming at the edge, developing the concept of *edge gaming*.

This research was supported by the Region of Madrid through the TAPIR-CM project (S2018/TCS-4496).

[1]https://telecoms.com/506387/mobile-cloud-to-drive-videogame-market-growth/

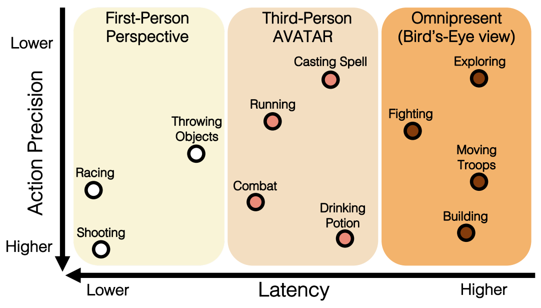

Fig. 1. High level example of latency requirements for different game actions.

In contrast to edge gaming, *i)* one problem of cloud gaming is to guarantee latency (depending on the type of game, 100-500 ms is the maximum latency conceded by users [3]–[5]) and bandwidth. Moving computing resources closer to end-users could decrease latency issues and thanks to 5G and multi-access point [1] dedicated bandwidth could be higher, increasing the overall Quality of Experience (QoE) for end-users. Fig. 1 shows a simple example on how different games require different latency. For instance, game actions such as shooting or driving are greatly affected by latency, while others, such as selecting avatars in turn-based strategic games are less sensitive. Edge gaming will enable in particular fast-paced games, where timing is fundamental (e.g. First Person Shooter (FPS)) and competitive online multiplayer gaming.

At the same time, *ii)* allocating games at the edge will reduce network core congestion [6], therefore reducing the possibilities of packet losses while allowing higher video quality. Providers could immediately sense what is happening in the network, reacting in a short time to dynamic latency conditions and bandwidth requirements (hence avoiding to ruin the gaming experience for end-users). Finally, *iii)* the paradigm could easily leverage new networking principles such as network slicing and standardization bodies such as ETSI MEC [1]. On the other hand, constrained edge resources are scarce and variable and new techniques should be proposed to efficiently provision resources to meet several Quality of Service (QoS) and QoE constraints. Bringing constrained capabilities to the edge rises new challenges (how to efficiently allocate resources) but also new costs (e.g. increased electricity bills) for providers. Noteworthy, there is a growing interest in making infrastructures more sustainable[2] and leveraging on intermittent renewable energies (e.g., solar and/or wind) to sustain a MEC network could help infrastructure providers

[2]https://www.un.org/sustainabledevelopment/infrastructure-industrialization/

to decrease overall costs. This also means that MEC nodes could have time-varying capacities according to intermittent renewable resources, rising the possibility to migrate resources across several edge nodes if necessary. Games could exploit migration at the edge because migration delays are negligible in such scenario due to the proximity of edge servers in residential areas [2] and new efficient migration techniques that can be exploited [7]–[9]. Therefore, by migrating online gaming servers in a few milliseconds would allow for seamless game sessions and therefore an increased QoE for end-users.

Infrastructure or services providers could leverage this new edge infrastructure to propose a novel MEC-based *premium* experience to interested end-users, willing to pay a higher fee but in exchange receiving a higher QoE with lower latency and higher image quality for their game sessions. Therefore, we focus on the maximization of utility in a *MEC-based green online gaming* scenario, which we refer to as **green edge gaming**. Accordingly, we develop a smart allocation and migration algorithm of online game sessions under several realistic constraints. We first allocate game sessions requests in the most suitable edge node and then migrate sessions according to the fluctuation of intermittent green energy.

A. Main Contribution

The main contributions of our paper are:

- We develop the concept of green edge gaming in a 5G and beyond 5G Edge setting with time-varying edge nodes capabilities, due to the time-varying availability of green energy. We show that this concept is compliant to ETSI MEC standardization and with modern networking principles such as network slicing.
- We argue that this concept could lead to a *premium* business scenario and therefore our goal is to maximise the service provider's utility, for which we formulate an accurate *multi*-dimensional linear integer programming problem, showing that it is NP-hard in the strong sense and sub-modular.
- We develop `GREENING`, an efficient online heuristic for game session allocation and migration with time-varying edge nodes capabilities; we allow for migrations in order to accept and fully serve as many requests as possible while minimizing the use of brown energy.
- Through simulations, we show that `GREENING` achieves near-optimal performance levels, which are at least 20% better than state-of-the-art approaches in terms of system utility, without requiring high complexity.

Paper Organization: The rest of this paper is organized as follows. In Section II we overview previous works, while in Section III we define the system model. In Section IV, we formulate an instantaneous optimization problem, proving its NP-hardness and submodularity. In Section V, we tackle the more general online problem of green game session allocation. Section VI highlights our main results and finally Section VII presents our concluding remarks.

II. Related Work

Cloud gaming has been well studied and is becoming a reality, with some projections highlighting that 20% of gaming sessions will be soon on cloud [10].

Compared to cloud gaming, edge gaming is a newer concept tied up with edge computing. Indeed, edge computing could help the cloud gaming paradigm in both storage [11] and computation (by offloading tasks [12] or rendering whole games), minimizing the overall response time. In [6], the authors leverage edge servers to offload computation intensive tasks for gaming, showing that this strategy could reduce network delay and bandwidth consumption. Yates *et al.* [13] develop a Markov model to optimize frame rate and lag synchronization of server and player in low-latency edge cloud gaming systems, employing an age of information metric to characterize the system performance. In an edge computing scenario with constrained computing resources migrations are necessary, therefore opening new scenarios with multiple challenges. Braun *et al.* [8] propose a new migration protocol to migrate a MEC gaming application through different edge servers while in [9], the authors developed Talaria, an in-engine content synchronisation solution. The latter allows for unnoticeable game instance migration between edge servers, which is mandatory in order to maintain a satisfactory QoE for end-users in fast-paced games.

Furthermore, allocation and migration of resources is a well-known problem, especially for centralized cloud systems [14]. Notwithstanding, it is difficult to apply the same strategies for an edge scenario, due to multi-constrained edge resources. Edge nodes could be constrained for instance in bandwidth, energy, storage or computing capabilities. For instance, [15] propose a joint service placement and request scheduling scheme, while [16] propose a randomized rounding technique for the joint optimization of service placement and request routing in a MEC network. Both papers consider several constraints on edge nodes. Other papers use a machine learning approach. Wang *et al.* [17] use Reinforcement Learning (RL) in a mobility-aware MEC network with the goal of obtain the maximum system utility minimizing the migration costs. In [18], the authors use a deep learning framework for proactive migration of MEC resources in a 5G vehicle scenario, with the goal of minimizing the total energy expenditure, without considering hardware limitations (such as on memory and CPU cycles). In [19], the authors leverage on deep reinforcement learning in order to minimize the average completion time of tasks under migration energy budget. Wang *et al.* [20], propose a Markov decision process framework for dynamic service migration in order to follow users movement, without considering edge nodes capabilities. In [21], the authors propose resource-aware VM migration technique, without considering energy, while on the opposite, the authors in [22] focused on a VM migration mechanism to leverage on fluctuating green energy, minimizing therefore the brown energy consumption, but without considering other constraints. No one so far considered a **green edge gaming** scenario, in which games

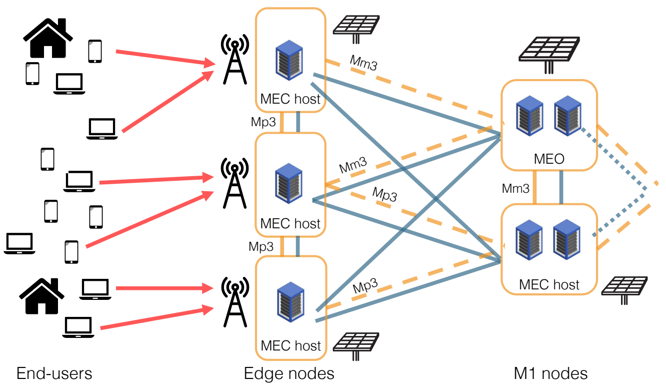

Fig. 2. 5G Edge Infrastructure compliant with ETSI MEC.

are allocated in nodes with time-varying capacity and in the presence of several nearby edge nodes. In this scenario, beside helping in meeting several network constraints fast job migration can not only reduce costs and pollution, but also avoid congestion in the network core, with improved QoE for end-users. Many works in the cloud gaming area have tackled QoS or QoE metrics (on delay and bandwidth especially), but did not consider all classes of constraints as we do. The reason is that, in a cloud infrastructure, many of these constraints are not challenging, due to abundance of resources, while they are important for the edge infrastructure. Also, we consider time-varying node capacity due to intermittent renewable energy, which has not been evaluated for edge/cloud gaming.

Most of previous works focused on a subset of our constraints and did not employ migration, or migration was mainly based on user's mobility, while we show that migration has a fundamental role independently of user mobility. We assume that migration of running jobs can be performed within nearby edge servers to either optimize the use of energy and resources or make space for new jobs while respecting all the constraints. This is possible only in the edge context, while in legacy cloud gaming contexts, migration cannot be considered at all [23].

III. System Model

Fig. 2 shows our reference scenario. We model a 5G edge network infrastructure [24] containing a set of edge game servers, with a set $\mathcal{Z}$ of network links in between. B of the servers reside on *edge nodes* deployed next to a different base station (BS). The rest of the servers reside each on a different *M1 node*, located deeper in the edge/transport network where the traffic of multiple BSs converges [24]. M1 nodes have bigger capacities compared to edge nodes, in terms of computation, energy and memory capabilities. All servers depend on green and brown energy. In our work, we focus only on wind and solar as renewable resources, since they have already been applied in edge computing contexts [25]. Brown energy is always available at M1 nodes, while edge nodes might not have access to it.

When they cannot use it, their capacity C_n is proportional to the available green power G_n. In all cases, we assume that green energy can be used at no cost, while brown energy has a non-negligible cost. G_n is modeled with a uniform distribution [26].

For simplicity, we do not consider the use of batteries at edge nodes, therefore green energy is not stored. Due to the unpredictability of wind/solar resources [27], we model the green energy behaviour in a stochastic manner.

The infrastructure is used to run game sessions, each of which is referred to as a *job*. We model a set $\mathcal{J}$ of jobs arriving over time, and the time is slotted. Jobs originate from mobile phones, laptops or smart TV. Thus, considering a large potential number of users, we consider that jobs arrive according to a Poisson process and have a duration extracted from a Weibull distribution, which realistically models the duration of online game sessions [28].

Each job j has a total revenue R_j, that considers many factors, e.g., mainly user's fees, but also percentages of game purchases, advertising, etc. Jobs require a random amount of computation, memory and energy, which determine their deployment cost ($C_j^{(d)}$, taken as a small portion of R_j) and brown energy cost ($C_j^{(b)}$, proportional to the use of brown energy in each time unit, hence this is not constant over time). Besides, migrations and interruptions of jobs incur penalty costs $C_j^{(m)}$ and $C_j^{(p)}$ proportionally to R_j. Specifically, migration is fast and cheap, so we consider that its cost is a small percentage of the migrating job's revenue. For simplicity, we do not consider migrations triggered by handovers as an optimization problem. We do so not only because that topic has been covered in other studies [8], but also because we are interested in the evaluation of interactive game sessions, which are typically several minutes long and are played at home or in a static environment [28]. A job interruption can occur when the availability of green energy decreases, the server's computing capacity becomes insufficient for all running jobs and migration cannot be enforced. The cost of interrupting a job is comparable with the job's revenue, because of the *premium* nature of the user's subscriptions. Accordingly, jobs have to be scheduled immediately or rejected rather than queued. Note that our approach to revenue and cost values does not consider topological factors such as the distance between nodes. In fact, those factors lead to negligible differences in the edge scenario [2].

Every job requires to meet QoS requirements in terms of delay and bandwidth. The overall response delay is the total time between an end-user submits his/her commands and the time the corresponding game frame is displayed to the user [29]. This includes network delay, processing delay, game logic and playout delay. For simplicity, we work with average delays and focus on the network delay budget of each job, D_j, considering the other delay components as constant.

Furthermore, we assume that queuing delays at switches are negligible, since our *premium* service could prioritize packets, avoiding unnecessary delays. The network delay budget is therefore spent over the links that connect the user to the game server, the resulting delay being the sum of average per-link delays d_z. This implies that we also neglect the migration time, since it is possible to obtain seamless game migration across several edge servers at millisecond timescale [9].

TABLE I
NOTATION USED IN THE PAPER

Notation	Meaning
$C_j, C_j^{(\tau)}$	Cost of job j and its per-slot cost
$C_j^{(b)}$	Brown energy cost (per used slot)
$C_j^{(d)}$	Deployment cost for job j
$C_j^{(m)}, C_j^{(p)}$	Migration and interruption costs for job j
$\mathcal{J}, J$	Set of jobs and its size
$\mathcal{N}, N$	Set of nodes (game servers), and its size
$R_j, R_j^{(\tau)}$	Revenue of job j and per-slot revenue
$\mathcal{T}$	Set of consecutive time slots
$\mathcal{Z}, Z$	Set of links and its size
t_j, T_n	Downlink throughput for job j and bandwidth of node n
d_z, D_j	Delay incurred on link z and budget delay of job j
e_j, G_n, E_n	Power needed by job j, and green/total power at node n
p_j, P_n	Computing power for job j and at node n
s_j, S_n	Memory required for job j and total memory at node n
w_{jz} = {0,1}	Placement variable telling if job j passes through link z
x_{jn} = {0,1}	Placement variable of job j at node n

Focusing instead on the bandwidth requirements, nowadays it is possible to play games in streaming with various screen resolutions, so we assume that each job requires a constant downlink bandwidth t_j, chosen at random from a uniform distribution. The uplink bandwidth is negligible [29]. Furthermore, since we focus on the edge environment, we assume that the downlink bandwidth of edge and M1 nodes is the bottleneck, so we ignore per-link bandwidth constraints.

We assume that per-time unit energy e_j, memory s_j and computing power p_j required by a job are known and constant during the whole job duration, with values drawn independently for each job from uniform distributions. In general, by considering fixed computing workloads and the use of resources for each job, we make a tractable simplification which makes sense to evaluate a system in which resources are always guaranteed to the user, hence they are allocated based on the peak demand of the online game session, which makes sense for a premium service like the one studied in this paper.

With the above, we next formulate a utility optimization problem on how to allot jobs to nodes so as to maximize the overall utility by serving as many jobs in full and minimizing total costs. This means that the use of green energy has to be prioritized, migrations should be used only if they bring more revenue than cost, and job interruptions should be avoided.

IV. Instantaneous Utility Optimization

First, we consider the instantaneous version of our problem, meaning that revenues and costs are allocated at each time slot, every job is allocated and executed in a single time slot and there are neither migrations nor job interruptions.

A. Problem Formulation

We consider the following variables: R_j is the revenue of accepted job j while C_j is its aggregated costs. C_j includes deployment $C_j^{(d)}$ and brown energy costs $C_j^{(b)}$ associated to the computation required for the job. x_{jn} is a binary decision variable: it is 1 if job j is allocated at edge node n, and 0 otherwise. w_{jz} is another binary variable, whose value is 1 if job j passes through link z and 0 otherwise. Table I summarizes the notation used in the paper. The problem is therefore formulated as follows:

$$\max \sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} (R_j - C_j)\, x_{jn}; \tag{1a}$$

$$\text{s.t.:} \sum_{n\in\mathcal{N}} x_{jn} \le 1, \qquad \forall j \in \mathcal{J}; \tag{1b}$$

$$\sum_{j\in\mathcal{J}} t_j x_{jn} \le T_n, \qquad \forall n \in \mathcal{N}; \tag{1c}$$

$$\sum_{j\in\mathcal{J}} p_j x_{jn} \le P_n, \qquad \forall n \in \mathcal{N}; \tag{1d}$$

$$\sum_{j\in\mathcal{J}} e_j x_{jn} \le E_n, \qquad \forall n \in \mathcal{N}; \tag{1e}$$

$$\sum_{j\in\mathcal{J}} s_j x_{jn} \le S_n, \qquad \forall n \in \mathcal{N}; \tag{1f}$$

$$\sum_{z\in\mathcal{Z}} d_z w_{jz} \le D_j, \qquad \forall j \in \mathcal{J}; \tag{1g}$$

where:

- The objective function (1a) expresses the net utility;
- Constraint (1b) tells that job j can only be allocated to one node n;
- Constraints (1c) to (1f) tell that a job's placement cannot violate the server's capacity in terms of: downlink bandwidth (T_n), processing power (P_n), available energy (instantaneous power E_n) and memory (S_n);
- Constraint (1g) ensures that the average delay is guaranteed for each job;
- All weights t_j, p_j, e_j, s_j, and d_z, capacities T_n, P_n, E_n, S_n, and delay budgets D_j take positive values.

The above described problem is non-trivial to solve if no server can accommodate all jobs. In that case, the problem is NP-Hard, as shown next.

Theorem 1. *Constraints* (1b) *and* (1c) *alone make the problem NP-hard (in the strong sense).*

Proof. We reduce the Multiple Knapsack Problem (MKP) to our problem formalization. According to [30], the MKP could be written as follows: considering a set of K knapsacks with capacity W_k each, $k \in \{1, \cdots, K\}$, and a set of I items to store ($K \le I$) where each item i has positive reward r_i and positive weight w_i, $i \in \{1, \cdots, I\}$. The total value is $\sum_{k=1}^{K} \sum_{i=1}^{I} r_i x_{ik}$, which has to be maximized under the constraints that $\sum_{i=1}^{I} w_i x_{ik} \le W_k, \forall k$, and $\sum_{k=1}^{K} x_{ik} \le 1, \forall i$, with x_{ik} being a binary variable indicating whether item i is allocated to knapsack k.

We take C_j equal to 0 or 1 and also consider the special case where the following variables of our problem, p_j, e_j, s_j, d_z are all equal to 1 and $P_n = J$, $E_n = J$, $S_n = J$, and $D_j = Z$. With this special configuration, our problem is a MKP with $K = N$ knapsacks of capacity T_n and $I = J$ items with weights t_j. This means that the MKP is a particular case of our problem. Therefore we could argue that our problem is complex as much as the MKP, which is NP-hard. Since this reduction can be built in polynomial time, it follows that our problem is NP-hard. However, we highlight that due to this reduction to MKP, our problem is NP-hard in the strong sense, meaning that no polynomial-time approximation scheme is known [30] unless $P = NP$. $\square$

B. Sub-modularity

We now show that the problem is sub-modular, which leads to nice performance guarantees. First, let's re-formulate the problem as a set-optimization problem. Let denote by $\mathcal{S} \subseteq \mathcal{J} \times \mathcal{N}$ the set of selected single-service placements, where $(j, n) \in \mathcal{S}$ means to place job j at node n. Let $\Theta(S)$ denote the objective value of (1a), so that (1) becomes as follows:

$$\max \Theta(S) \tag{2a}$$
$$\text{s.t.: } \mathcal{S} \subseteq \mathcal{J} \times \mathcal{N} \tag{2b}$$
$$(1b) \text{ to } (1g). \tag{2c}$$

Theorem 2. *The optimal value of Θ is a monotone increasing and sub-modular set function.*

Proof. Consider that a real-valued set function f is monotone increasing if $\forall\, \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}$, $f(\mathcal{S}_1) \le f(\mathcal{S}_2)$. Moreover, the function $f(\cdot)$ is sub-modular if $\forall\, \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}$ and $u \in \mathcal{S} \setminus \mathcal{S}_2$, it holds that $f(\{u\} \cup \mathcal{S}_1) - f(\mathcal{S}_1) \ge f(\{u\} \cup \mathcal{S}_2) - f(\mathcal{S}_2)$.

The monotonicity of the solution of our problem is clear because expanding $\mathcal{S}$ (i.e., putting more jobs and/or nodes) enlarges the solution space of (2a) and therefore increases its optimal value. The solution is also sub-modular, since the more jobs we allocate, the more brown energy nodes will have to use, and therefore the overall utility obtained by including more jobs will be progressively reduced. For this class of problems, it is known that we can construct a greedy algorithm that iteratively selects an element, subject to constraints, that maximizes the objective function, and such algorithm achieves a performance guarantee of $1 - 1/e$ [31]. $\square$

Algorithm 1 shows the legacy GREEDY algorithm that solves problem (2) in polynomial time with performance guarantees using its submodularity property. The structure of the algorithm derives from [31], so we omit to comment on it.

Algorithm 1 GREEDY

```
1:  Input: Network topology, $\mathcal{N}$, jobs $\mathcal{J}$ (with parameters $t_j, p_j, s_j, e_j, D_j\, \forall j \in \mathcal{J}), T_n, P_n, S_n, G_n, E_n\, \forall n \in \mathcal{N}, d_z\, \forall z \in \mathcal{Z}$.
2:  Output: Job-to-node placement map $S$
3:  $\mathcal{S} = \emptyset$
4:  $\mathcal{J}^\star = J$
5:  $\mathcal{S}^\star = \mathcal{J} \times \mathcal{N}$
6:  while $\exists (j, n) \in \mathcal{S}^\star$ s.t. $\mathcal{S} \cup (j, n)$ satisfies (2c) do
7:      $(j^\star, n^\star) \leftarrow \arg\max_{(j,n) \in \mathcal{S}^\star} \mathcal{S} \cup (j, n)$
8:      $\mathcal{S} \leftarrow \mathcal{S} \cup (j^\star, n^\star)$
9:      $\mathcal{J}^\star \leftarrow \mathcal{J}^\star \setminus j^\star$
10:     $\mathcal{S}^\star = \mathcal{J}^\star \times \mathcal{N}$
11: end while
```

V. Online Decisions with Migrations and Penalties

Our problem can also be extended to the case of jobs that last more than a slot and arrive asynchronously. This case is important because it represents the practical problem to be solved online in a real system. The objective of the optimization becomes as follows:

$$\max \sum_{\tau \in \mathcal{T}} \sum_{n \in \mathcal{N}} \sum_{j \in \mathcal{J}} \left(R_j^{(\tau)} - C_j^{(\tau)} \right) x_{jn}^{(\tau)} \tag{3}$$

where $\mathcal{T}$ is the time interval (a set of consecutive slots) over which to optimize. In this case, the revenue $R_j^{(\tau)}$ is positive at job acceptance time and zero otherwise. Similarly, the cost $C_j^{(\tau)}$ has to include the deployment cost ($C_j^{(d)}$, only at acceptance time), the variable cost of using brown energy ($C_j^{(b)}$, in each used slot), and the penalties for migration ($C_j^{(m)}$) and for job interruption ($C_j^{(p)}$) in the time slots in which they occur. The constraints to satisfy are like in (1b)–(1g), except for the fact that constraints have to hold at any time slot $\tau \in \mathcal{T}$.

In this new formulation, jobs can arrive in different time slots, and decisions must be made online at the slot boundaries. Noteworthy, if migration and job interruption costs are neglected, the problem is equivalent to the one shown in Section IV, because it is enough to maximize the objective function slot by slot. Therefore the sub-modularity property will hold also for the online job allocation problem under such simplifying assumptions. Instead, if we consider penalties, sub-modularity is not guaranteed. However, with realistically small migration costs and rare job interruptions, the problem can be considered, *in practice*, still sub-modular (or as a small perturbation of a sub-modular case). Hence the intuition that we can extend the greedy heuristic approach also to the online version of the problem, as shown in what follows.

A. Online Heuristic

With a sub-modular problem, we can consider that when a job arrives, we only need to maximise the instantaneous utility. Therefore we do not need to reject any job just because that might prevent the acceptance of future jobs with higher revenue. However, this does not mean that we need to use the GREEDY algorithm. Instead, while we retain the spirit of the GREEDY algorithm, we propose to base the instantaneous job allocation decision on the status of servers and the available green energy.

Our proposal, named GREENING, is described in Algorithm 2. At the beginning of each time slot, the heuristic sorts all nodes according to their green energy level, in descending order. This is motivated by the fact that nodes that have more green energy can have more computing power and incur less costs. Indeed, this approach is in line with the fact that we are interested in minimizing both the cost for the use of brown energy and job interruptions.

First, the GREENING algorithm checks whether the level of green energy of any node had changed. If it were so, some nodes might no longer have enough power to serve all their allocated jobs, so that some jobs should be migrated or interrupted. The algorithm proceeds node by node, migrating

Algorithm 2 GREENING (at each slot τ)

```
1: Input: topology, $\mathcal{N}$, active ($\mathcal{J}^{(\tau)}$) and new ($\mathcal{J}^{(+)}$) jobs with parameters $t_j, p_j, s_j, e_j, D_j \, \forall j \in \mathcal{J}^{(\tau)} \cup \mathcal{J}^{(+)}$, network parameters $T_n, P_n, S_n, E_n^{(\tau)}, E_n^{(\tau-1)} \, \forall n \in \mathcal{N}, d_z \, \forall z \in \mathcal{Z}$, and previous allocation $\mathcal{S}^{(\tau-1)}$.
2: Output: $\mathcal{S}^{(\tau)}, \Theta^{(\tau)}$
3: Initialize: $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau-1)}; \Theta^{(\tau)} \leftarrow 0; \mathcal{J}^{(a)} \leftarrow \emptyset$
4: $\mathcal{N}^{(\tau)} \leftarrow \texttt{sort}(\mathcal{N}, G_n^{(\tau)})$
5: for $n \in \mathcal{N}$ do
6:   if $E_n^{(\tau)} < E_n^{(\tau-1)}$ then
7:     while $\sum_{j \in \mathcal{J}} x_{jn} e_j > E_n^{(\tau)}$ do
8:       $n^{(m)} \leftarrow \texttt{Migr}(j^{(m)} \leftarrow \arg\min_j \{C_j^{(m)} | x_{jn} = 1\})$
9:       if $n^{(m)} == -1$ {the job is interrupted} then
10:        $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \setminus \{j^{(m)}\}$
11:        $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - C_{j^{(m)}}^{(p)}$
12:        $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau)} \setminus \{(j^{(m)}, n)\}$
13:      else if $n^{(m)} \neq n$ {the job is migrated} then
14:        $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - C_{j^{(m)}}^{(m)}$
15:        $\mathcal{S}^{(\tau)} \leftarrow \left(\mathcal{S}^{(\tau)} \setminus \{(j^{(m)}, n)\}\right) \cup \{(j^{(m)}, n^{(m)})\}$
16:      end if
17:    end while
18:  end if
19: end for
20: for $j \in \mathcal{J}^{(+)}$ do
21:   for $n \in \mathcal{N}^{(\tau)}$ do
22:     if $(j, n)$ satisfies (1c)–(1g) then
23:       $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \cup \{j\}$
24:       $\mathcal{S}^{(\tau)} \leftarrow \mathcal{S}^{(\tau)} \cup \{(j, n)\}$
25:       $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} + R_j - C_j^{(d)}$
26:       break
27:     end if
28:   end for
29:   if $\nexists n \in \mathcal{N}$ s.t. $(j, n) \in \mathcal{S}^{(\tau)}$ then
30:     $\mathcal{J}^{\star} \leftarrow \mathcal{J}^{(\tau)} \cup \mathcal{J}^{(a)}$
31:     while $\mathcal{J}^{\star} \neq \emptyset$ do
32:       $j^{(m)} = \arg\min_{j \in \mathcal{J}^{\star}} \{C_j^{(m)}\}$
33:       $n^{(m)} = n$ s.t. $(j^{(m)}, n) \in \mathcal{S}^{(\tau)}$
34:       $\mathcal{J}^{\star} \leftarrow \mathcal{J}^{\star} \setminus \{j^{(m)}\}$
35:       if replacing $j^{(m)}$ with $j$ satisfies (1c)–(1g) then
36:         $n^{(m\prime)} = \texttt{Migr}(j^{(m)})$
37:         if $n^{(m\prime)} \neq -1 \,\&\, n^{(m\prime)} \neq n^{(m)}$ then
38:           $\mathcal{J}^{(\tau)} \leftarrow \mathcal{J}^{(\tau)} \cup \{j\}$
39:           $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - C_{j^{(m)}}^{(m)} + R_j - C_j^{(d)}$
40:           $\mathcal{S}^{(\tau)} \leftarrow \left(\mathcal{S}^{(\tau)} \setminus \{(j^{(m)}, n^{(m)})\}\right) \cup \{(j^{(m)}, n^{(m\prime)})\} \cup \{(j, n^{(m)})\}$
41:           $\mathcal{J}^{\star} \leftarrow \emptyset$ {migration and allocation completed}
42:         end if
43:       end if
44:     end while
45:   end if
46: end for
47: $\Theta^{(\tau)} \leftarrow \Theta^{(\tau)} - \sum_{(j,n) \in \mathcal{S}^{(\tau)}} C_j^{(b)}$
```

jobs in increasing migration cost order, until the capacity of the node is enough for the remaining jobs. If a job cannot be migrated, then it is interrupted. The `Migr` function (Algorithm 3) is in charge of checking to which node a specific job can be migrated (it returns -1 if no node can host the job).

Second, `GREENING` goes over the list of newly arrived jobs and tries to allocate them one by one, starting from the node with the highest available green power, which is the cheapest solution. The algorithm tries a direct placement on the node at the top of the list, and moves to the next node only if the allocation is not possible according to any of the constraints. This is aligned with the greedy heuristic of the instantaneous problem, although it simply looks at energy levels rather than overall allocation utilities. The probability that the proposed scheme makes the same decision of the greedy approach is however high, because the node with the highest unused green energy level is likely the one that offers the highest utility.

However, if no node in the list can take a newly arrived job, `GREENING` tries to migrate other already allocated jobs to make space for the new job. This part of the algorithm substantially differs from a standard greedy heuristic. This is done by invoking again the migration function `Migr` on the already allocated jobs, starting from the one with the lowest migration cost. If the migration function does not find any migration combination that makes enough room for the new job, the job is dropped, and its revenue is lost, otherwise the job is allocated, with a revenue R_j and the migration is committed, which incurs a cost $C_j^{(m)}$.

Eventually, the algorithm discounts the cost of used brown energy in that slot from the objective function.

Algorithm 3 shows that the migration function operates a simple search on the set of potential migration destination nodes and checks the feasibility of migration based on the problem's constraints. The search goes from the node with more green energy to the one with the least green energy, and stops as soon as a destination node is found. Algorithm 3 returns the destination node identifier, or -1 in case no migration is possible and the node that currently hosts the job does not have enough power for all its allocated jobs. Note that the described migration function is greedy and so Algorithm 2 is still a greedy algorithm, in the sense that it makes instantaneous decisions without considering what could happen in the future. However, using the migration function can only improve the utility obtained with a scheme without migration, be it Algorithm 1 or Algorithm 2 simplified by skipping the call to the migration function. Therefore, we can expect that Algorithm 2 will offer better performance guarantees than the value $1 - 1/e$ of Algorithm 1.

To conclude, the complexity of our `GREENING` heuristic described in Algorithm 2 is $O(N^2 J^2)$, which would reduce to $O(NJ^2)$ in case of direct placement of the arriving jobs, without migrations.

B. ETSI MEC and network slicing compatibility

In this subsection we comment on how green edge gaming is compliant to both ETSI MEC and network slicing concepts.

Algorithm 3 `Migr`

```
1:  **Input:** $j^{(m)}$ (job to migrate)
2:  **Inherit:** State and variables of Algorithm 2
3:  **Output:** $n^{(m)}$ (node where to migrate)
4:  $n^{(m)} \leftarrow n$ s.t. $(j^{(m)}, n) \in \mathcal{S}^{(\tau)}$
5:  **for** $n' \in \mathcal{N}^{(\tau)} \setminus \{n^{(m)}\}$ **do**
6:      **if** allocating $j^{(m)}$ to $n'$ satisfies (1c)–(1g) **then**
7:          $n^{(m)} \leftarrow n'$
8:          break
9:      **end if**
10: **end for**
11: **if** $n^{(m)} == n$ s.t. $(j^{(m)}, n) \in \mathcal{S}^{(\tau)}$ **then**
12:     **if** $\sum_{j \in \mathcal{J}^{(\tau)} \cup \mathcal{J}^{(a)}} x_{jn} e_j > E_n$ **then**
13:         $n^{(m)} \leftarrow -1$
14:     **end if**
15: **end if**
```

In the case of ETSI MEC, the MEC Orchestrator (MEO), which has an overview of the complete MEC system and therefore could be deployed in more centralized nodes, could consider the objective function (1a) to place games in its system. Indeed, one of the MEO's roles consists in selecting appropriate MEC host(s) for application instantiation based on constraints, such as latency, available resources, and available services [1]. To this aim, the MEO talks directly to the Virtual Infrastructure Manager (VIM), whose role is to physically deploy resources. MEC hosts provide compute, storage and network resources for the MEC applications and they could be deployed in edge nodes/M1 nodes, where games are actually installed. Finally, games could be deployed as MEC applications, leveraging on several *on-board* MEC services, such as Radio Network Information, location and traffic management to sustain the appropriate QoE level. Edge and M1 nodes could be connected through several reference points: with the MEO through a *Mm3* link and they could communicate between each other through a *Mp3* link [1]. Indeed, Fig. 2 shows also a high level example of the edge gaming scenario implemented through ETSI MEC.

Green edge gaming could also leverage network slicing to guarantee resources to servers. A provider could reserve a slice of resource, in order to satisfy end-users in terms of bandwidth computing power.

VI. Evaluation results

In this section we provide a numerical evaluation of our proposal on green edge gaming and a performance comparison to alternative online gaming solutions. To perform our experiments we built a simulator with Matlab 2021a, in which we implemented our solution as well as a few baselines and state-of-the-art alternatives.

A. Simulation scenario and setup

We study a metropolitan area where users leverage online game servers in edge and M1 nodes, with the QoS requirements described before in terms of computing power, latency, memory and bandwidth. The number of nodes N is fixed in each simulated scenario, and we test N from 5 to 15, which is a range compatible with the number of edge and M1 nodes that will be initially deployed in a metropolitan framework, and is in line with previous studies [24]. In particular, we test three configurations: $N = \{5, 11, 15\}$, out of which 2, 5 and 7 are M1 nodes, respectively. The network topology is hierarchical, as displayed in Figure 2, and the connectivity in between servers is a full mesh. Servers capabilities are based on a NVidia blade server for edge computing and are 3.5 GHz for computing power, 64 GB for memory and using 250 W per blade, with 3 blades on edge node and 5 blades on M1 nodes. Besides, we assume that bandwidth and delays are constant and inline with 5G values (350 Mb/s for downlink and 5-10 ms for latency). The number of arrivals in each slot is given by a Poisson random variable with rate $\lambda = 0.2N$, hence, in each experiment we fix the value of λ to a value which can range from 1 to 3 arrivals/slot. The duration of a job is extracted from a Weibull distribution with parameter $k = 0.9637$ and $\mu = 2504.8$, which yield an average of about 40 minutes for a typical online game session [28], while the bandwidth can vary uniformly from 10 to 35 Mb/s for video resolutions that range from 720p for 4K.[3] Other parameters (computing, memory, power) are inline with previous works [10], [32] and values are as well extracted from uniform distributions. For instance, the computing power of a job ranges from 315 to 385 Mcycles/s (10 to 15% of a decent CPU core), and the absorbed power is 25 to 35 W. With the above numbers and a system working at full node capacity, on average, an edge node could serve a maximum of about 27 jobs at the same time, while a M1 node can serve up to about 45 jobs, on average. We solve problem (3) on slot-by-slot basis in order to simulate a typical day. Specifically, one time slot is equivalent to one minute in real life, which is much less than a typical online game session ($\sim$40 minutes) and is much longer than any migration (lasting tens of milliseconds [9]) or game session launching times, which take less than a second [7]. Every 15 minutes, we randomly change green energy levels without space-time correlation, which is inline with previous works [27]. We use a uniform distribution ranging from 0 to 60% of an M1 node power (i.e., from 0 to $0.6 \times 5 \times 250 = 750$ W). This range gives the opportunity to have edge nodes completely depending on green energy (note that the power required to run 3 NVidia blades at an edge node is 750 W). We extract the revenue R_j of a job j from a uniform distribution ranging from 0.03\$ to 0.0367\$ (to have an average of 10 cents every 3 jobs). The migration cost of a job is 5% of its revenue R_j while the deployment cost is just 1% of R_j. The brown energy cost is 0.2\$ per kWh, which is a realistic value. In case of job interruption, the penalty is a monetary cost, in particular the 95% of the interrupted job's revenue R_j. This is appropriate for a premium service scenario like the one considered in this work. In our experiments, the main performance metric is the system utility, which is computed on a per time slot basis. The average system utility is proportional to the objective function of our online optimization problem (3), so that it represents the performance

[3]https://stadia.google.com

of the tested algorithms. To evaluate the user's QoE, we show the *completion time*, which we define as the ratio between the sum of the actual aggregated activity duration of accepted jobs (including interrupted jobs, for the time they are active) over the total aggregate nominal duration of all accepted jobs. We also evaluate the service QoS by means of the number of jobs in the system as well as rejected and interrupted jobs. Other typical QoS metrics like jitter or packet losses are less relevant in the edge gaming scenario considered here, in which their values are typically small, so we omit their study. The above metrics are computed in terms of average and 95% confidence intervals for a number of alternative algorithms:

- `GREENING` is our heuristic defined in Algorithm 2.
- `Baseline` is our heuristic in which we disable the migration function.
- `Solver` is the algorithm that uses a Matlab *intlinprog* solver for the integer linear program (3), evaluated at each time slot over a time horizon of one time slot ($|\mathcal{T}| = 1$). The solution could not always be optimal since we put a maximum evaluation time of 1 minute for each time slot, in order to avoid long lasting experiments when the number of nodes exceeds a few units (11 or more in our settings, for which we needed up to one week to complete one experiment set notwithstanding the imposed timeout).
- `Future-1` is a baseline taken from [21], which is a placement algorithm with migration. Since the authors did not consider power, in this version we added our power constraint in their algorithm.
- `Future-2` is the original placement algorithm defined in [21], without energy constraints.
- `Random`, `Total green energy` and `Partial green energy` are instead algorithms with probabilistic placement and do not consider migrations. The first one considers a random job placement (all nodes have equal probability to be chosen), the second one gives probabilities proportionally to the presence of green power. The last one assigns probabilities to nodes proportionally to the level of unused green power.

B. Results

Fig. 3 shows the average utilities per time slot when edge nodes have static capacities, i.e., they can use brown power when run out of green power. The load used is proportional to the number of nodes, i.e., we use $\lambda = 0.2N$, which corresponds to a moderate utilization of edge game resources (about 50%–60% of the total available computing resources). We can see that most of the algorithms perform at the same level, with `GREENING` being slightly superior. This occurs because the revenue deriving from accepting a job is greater than its cost, so that eventually all algorithms tend to maximize the amount of accepted jobs. They do that by means of direct placement, with migration only used when necessary. However, when edge nodes con only use green energy, the story changes, especially worsening the performance of algorithms that cannot enforce migrations. Fig. 4 shows the average utility per time slot for a scenario like for Fig. 3, except now the capacity of edge nodes becomes time-varying due to its tie to the availability of green energy. Indeed, from now on, all figures will consider edge nodes which fully depend on green energy. In Fig. 4, we can see that `GREENING` achieves almost the same utility as `Solver`, and largely outperforms all other algorithms (at least by a 20%). The situation does not change significantly with higher or lower values of λ, so we omit to show the corresponding results here. However, note that to obtain the values reported in the figures above, `Solver` takes several days on a Dell T640 server with 128 GB of RAM and 40 logical cores with a variable clock rate (but *intlingprog* uses only 1 thread per instance, so we parallelized the number of experiment replicas rather than the single experiment), while other algorithms need just a few minutes. The second best heuristics are `Future-1` and `Future-2`, although their performance levels diminish as the number of game servers increases. Afterwards, it is the turn of `Baseline`, which always places jobs in the node with most green energy but does not have a migration function. Finally, there are all the heuristics with a probabilistic allocation approach (`Total green energy`, `Partial green energy` and `Random`), with the latter performing worse due to its randomness.

Fig. 5 shows the distance between the average utility and a simple utility upper bound. The latter is evaluated by multiplying the average revenue of a job, minus its deployment cost, times the average number of jobs arriving in a time slot. We can see from the figure that, with `Solver` and `GREENING`, this difference decreases with the number of nodes, meaning that such algorithms are progressively more efficient and asymptotically optimal (`GREENING` is just a few percents off the bound). The efficiency of other algorithms is instead largely insensitive to the number of nodes. As an interesting note, since `GREENING` and `Solver` are so close to the lower bound, we can argue that our bound must be tight with respect to the optimal. The bound is valid not only for greedy allocations, but it is valid in general over any time horizon and scheduling of jobs (including for jobs delayed before being deployed), because the bound considers the overall job arrival rate, not just the accepted jobs. Therefore we must conclude that `GREENING` and `Solver` are near-optimal, and in particular achieve much better performance than what guaranteed by using any standard greedy algorithm, which cannot guarantee anything better than $1-1/e \simeq 63\%$ of the optimal.

The superiority of `GREENING` is also visible by observing the average number of jobs active in the system, which is depicted in Fig. 6. Against intuition, high numbers of active jobs do not necessarily mean more rejected jobs. Indeed, Fig. 7 shows that in all scenarios, `Solver` and `GREENING` reject less jobs than `Future-1`, `Future-2` and `Baseline`. This occurs because `Solver` and `GREENING` are able to migrate more successfully allocated jobs in different nodes and therefore making enough room to accept newly arrived jobs. In any case, it is convenient to reject some jobs rather than interrupt them. Instead, other heuristics accept more jobs, but then they are forced to abruptly interrupt many jobs, which explains the fact that `Future-1`, `Future-2` and `Baseline` eventually

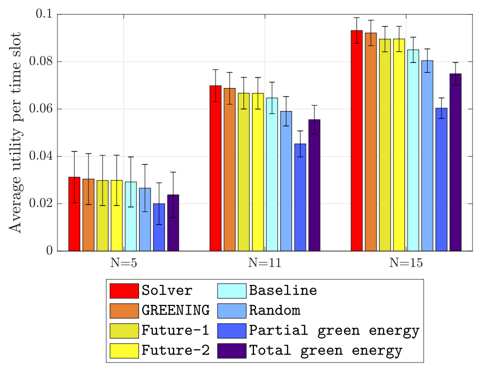

Fig. 3. Utility per time slot with edge nodes static capacities. $\lambda = 0.2N$.

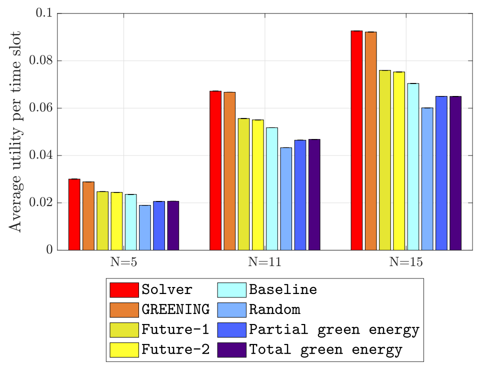

Fig. 4. Utility per time slot with time-varying nodes capacities. $\lambda = 0.2N$.

sustain less active jobs, on average, and experience lower utilities.

The figure also shows that probabilistic allocation algorithms reject much more jobs, which is due to the fact that they can allocate jobs to nodes that are close to saturation.

To evaluate the impact of job interruptions, Fig. 8 shows the completion time. The figure shows that GREENING achieves results close to 100%, which means that jobs start and almost always complete. Probabilistic algorithms and baselines incur high penalties and only manage to provide resources for 60% to 80% of the time for which servers commit upon accepting jobs. In this figure in particular, it is possible to see how targeted migrations are important in a dynamic and constrained scenario such as the edge of the network.

This confirms that GREENING is structured in a very efficient and robust way thanks to the use of migrations, without which the utility improvements due to the use of green energy could not be harvested, as clearly visible by comparing GREENING, Baseline and Total green energy. We conclude by remarking that all the above discussed figures show that not only GREENING goes well beyond the $1-1/e$ performance guarantee of a pure greedy approach, but also that GREENING performs near-optimally, especially as the number of game servers grows, and without requiring the complexity of a proper solver.

VII. Conclusion

In this paper, we have studied the green edge gaming concept. Arguing that network providers could offer such a service as a

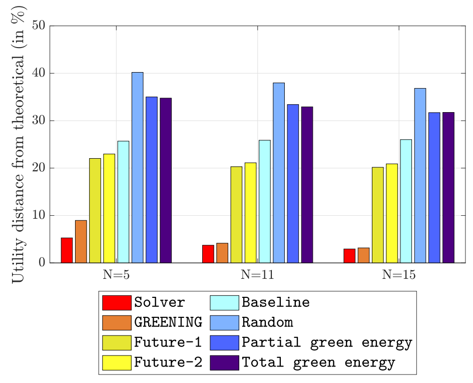

Fig. 5. Average utility distance from theoretical bound. $\lambda = 0.2N$.

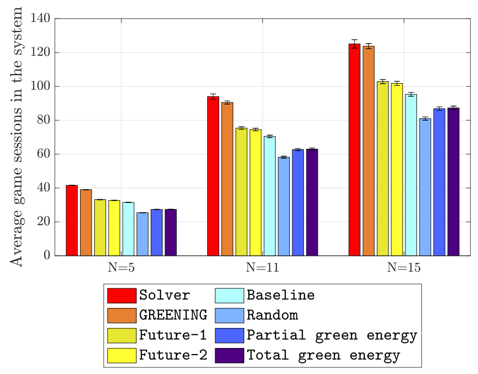

Fig. 6. Average active jobs in the system per time slot.

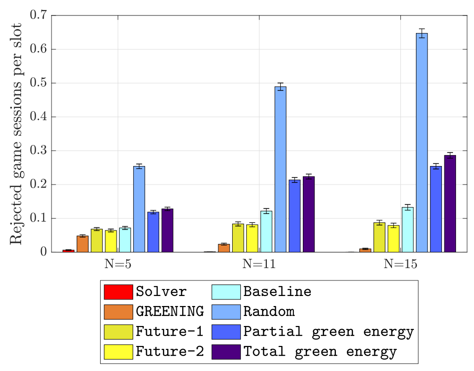

Fig. 7. Average rejected jobs per time slot. $\lambda = 0.2N$.

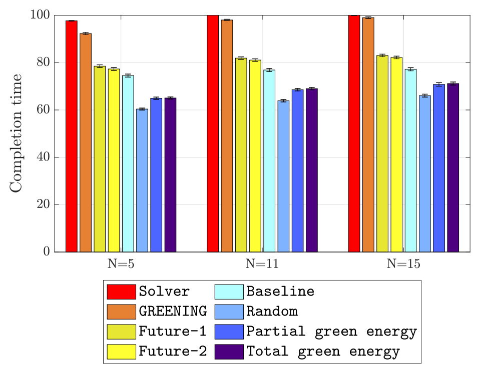

Fig. 8. Percentage of successfully completed game sessions.

premium service, we have studied how to maximize their utility by leveraging edge computing facilities of modern cellular networks and the possibility to partially feed online game servers with locally generated green energy.

We have formulated a *multi*-constrained integer-linear problem for the one-shot allocation of game sessions to servers, and shown that is it NP-hard in strong sense. The problem is however sub-modular, which encourages the use of greedy heuristics with strict performance guarantees.

We have therefore shown that the extension of the problem to an online version with game session migrations and interruptions is also tractable with an energy-based greedy approach. In particular, we have shown that the green energy component is key to drive the optimization of job allocations and migrations. Our proposed algorithm, `GREENING` largely outperforms state-of-the-art approaches and achieves near-optimal results with very low complexity. Notably, without the possibility to timely migrate online game sessions, which is possible only in an edge context, the greening of online gaming could not be a viable solution for providers, as we have numerically shown.

References

[1] ETSI, "Multi-Access Edge Computing (MEC); Framework and Reference Architecture," ETSI MEC ISG, Tech. Rep., jan 2019.

[2] F. Spinelli and V. Mancuso, "Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility," *IEEE Communications Surveys Tutorials*, vol. 23, no. 1, pp. 596–630, 2021.

[3] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users' perspective," *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2883–2894, 2013, Information System Security and Performance Modeling and Simulation for Future Mobile Networks.

[4] L. Pantel and L. C. Wolf, "On the Impact of Delay on Real-Time Multiplayer Games," in *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 23–29.

[5] M. Claypool and K. Claypool, "Latency and Player Actions in Online Games," *Commun. ACM*, vol. 49, no. 11, p. 40–45, nov 2006.

[6] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, "Improving Cloud Gaming Experience through Mobile Edge Computing," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 178–183, 2019.

[7] T. Kämäräinen, Y. Shan, M. Siekkinen, and A. Ylä-Jääski, "Virtual machines vs. containers in cloud gaming systems," in *2015 International Workshop on Network and Systems Support for Games (NetGames)*, 2015, pp. 1–6.

[8] P. J. Braun, S. Pandi, R.-S. Schmoll, and F. H. P. Fitzek, "On the study and deployment of mobile edge cloud for tactile Internet using a 5G gaming application," in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 154–159.

[9] T. Braud, A. Alhilal, and P. Hui, "Talaria: In-engine synchronisation for seamless migration of mobile edge gaming instances," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 375–381.

[10] E. Mills, N. Bourassa, L. Rainer, J. Mai, A. Shehabi, and N. Mills, "Toward Greener Gaming: Estimating National Energy Use and Energy Efficiency Potential," *The Computer Games Journal*, vol. 8, no. 3, pp. 157–178, 2019.

[11] K. Bilal and A. Erbad, "Edge computing for interactive media and video streaming," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 68–73.

[12] L. Lin, X. Liao, H. Jin, and P. Li, "Computation Offloading Toward Edge Computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[13] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[14] K. Li, H. Zheng, and J. Wu, "Migration-based virtual machine placement in cloud systems," in *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, 2013, pp. 83–90.

[15] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service Placement and Request Scheduling for Data-Intensive Applications in Edge Clouds," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 779–792, 2021.

[16] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service Placement and Request Routing in MEC Networks With Storage, Computation, and Communication Constraints," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[17] D. Wang, X. Tian, H. Cui, and Z. Liu, "Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware MEC network," *China Communications*, vol. 17, no. 8, pp. 31–44, 2020.

[18] I. Labriji, F. Meneghello, D. Cecchinato, S. Sesia, E. Perraud, E. C. Strinati, and M. Rossi, "Mobility Aware and Dynamic Migration of MEC Services for the Internet of Vehicles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 570–584, 2021.

[19] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed Task Migration Optimization in MEC by Extending Multi-Agent Deep Reinforcement Learning Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1603–1614, 2021.

[20] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic Service Migration in Mobile Edge Computing Based on Markov Decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[21] G. J. L. Paulraj, S. A. J. Francis, D. Peter, and I. J. Jebadurai, "Resource-aware virtual machine migration in IoT cloud," *Future Generation Computer Systems*, vol. 85, pp. 173–183, 2018.

[22] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, and W. Dai, "Energy efficient task allocation and energy scheduling in green energy powered edge computing," *Future Generation Computer Systems*, vol. 95, pp. 89–99, 2019.

[23] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Placing Virtual Machines to Optimize Cloud Gaming Experience," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 42–53, 2015.

[24] ITU-T, "Consideration on 5G transport network reference architecture and bandwidth requirements," ITU-T Study Group, Tech. Rep., feb 2018.

[25] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya, "On Enabling Sustainable Edge Computing with Renewable Energy Resources," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 94–101, 2018.

[26] D. Renga and M. Meo, "Dimensioning Renewable Energy Systems to Power Mobile Networks," *IEEE Transactions on Green Communications and Networking*, vol. 3, no. 2, pp. 366–380, 2019.

[27] Y. Mao, Y. Luo, J. Zhang, and K. B. Letaief, "Energy harvesting small cell networks: feasibility, deployment, and operation," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 94–101, 2015.

[28] D. Finkel, M. Claypool, S. Jaffe, T. Nguyen, and B. Stephen, "Assignment of games to servers in the OnLive cloud game system," in *2014 13th Annual Workshop on Network and Systems Support for Games*, 2014, pp. 1–3.

[29] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, and C.-H. Hsu, "On the Quality of Service of Cloud Gaming Systems," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 480–495, 2014.

[30] S. Martello, "Knapsack Problems: Algorithms and Computer Implementations," *Wiley-Interscience series in discrete mathematics and optimiza tion*, 1990.

[31] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, *An analysis of approximations for maximizing submodular set functions—I.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 265–294.

[32] Y. Zhang, P. Qu, J. Cihang, and W. Zheng, "A Cloud Gaming System Based on User-Level Virtualization and Its Resource Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1239–1252, 2016.