Energy-Delay Tradeoff for Virtual Machine Placement in Virtualized Multi-Access Edge Computing: A Two-Sided Matching Approach

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

05-12-2020 / 25-01-2021

CITATION

Zhang, Long; Zhuge, Shanshan; Wang, Yao; Xu, Haitao; Sun, Enchang (2020): Energy-Delay Tradeoff for Virtual Machine Placement in Virtualized Multi-Access Edge Computing: A Two-Sided Matching Approach. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.13336622.v2>

DOI

[10.36227/techrxiv.13336622.v2](https://doi.org/10.36227/techrxiv.13336622.v2)

Energy-Delay Tradeoff for Virtual Machine Placement in Virtualized Multi-Access Edge Computing: A Two-Sided Matching Approach

Long Zhang, Shanshan Zhuge, Yao Wang, Haitao Xu, and Enchang Sun

Abstract

By decoupling network functions from the underlying physical machines (PMs) at the edge of the networks, the virtualized multi-access edge computing (MEC) enables deployment of new network services and elastic network scaling to reduce maintenance costs in a more flexible, scalable and cost-effective manner. Although there are appealing performance gains to be achieved, the placement of virtual machines (VMs) on top of the sharing PMs to support computation-intensive applications for the smart mobile devices becomes a major challenge, especially for an increasing network scale. In this paper, we attempt to deal with the VM placement problem in virtualized MEC system, which is targeted for finding a performance balance between energy consumption and computing/offloading delay. To capture such a tradeoff for VM placement, we formulate a weighted sum based cost minimization problem as a pure 0-1 integer linear programming problem, which is NP-complete and very difficult to solve with lower complexity. Based on the one-to-one mapping relation constraint, the VM placement problem is converted into a many-to-many two-sided matching problem between the VM instances and the PMs. Motivated by the student project allocation problem, we develop an extended two-sided matching algorithm with lower computational complexity for solving the many-to-many matching problem. Simulation results are presented to demonstrate the effectiveness of our proposed matching algorithm, and the normalization factor is of great significance to obtain lower total cost.

Index Terms

Multi-access edge computing, network functions virtualization, virtual machine placement, physical machine, energy-delay tradeoff, matching theory.

L. Zhang, S. Zhuge, and Y. Wang are with the School of Information and Electrical Engineering, Hebei University of Engineering, Handan 056038, China (e-mail: zhanglong@hebeu.edu.cn; zhugess1021@aliyun.com; yaow0518@gmail.com).

H. Xu is with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: xuhaitao@ustb.edu.cn).

E. Sun is with the Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology, Beijing 100124, China, and also with the Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China (e-mail: ecsun@bjut.edu.cn)

1. Introduction

Nowadays, along with the technological evolution of the fifth generation (5G)-based cellular networks and the Internet of Things (IoT), the growing popularity of smart mobile devices (SMDs) has been driving the rapid development of various attractive mobile applications and multimedia services, such as extended reality, holographic telepresence, cloud gaming, autonomous driving, eHealth, etc [1]. These newly emerging applications and services heavily rely on high-speed data rates and low-latency transmission, which bring a big challenge to the computation capabilities, storage resources, and energy consumptions of the SMDs. Therefore, it is a very challenging task to run the computation-intensive applications on the resource-hungry SMDs while meeting strict delay requirements. The tension between the low capabilities of the SMDs and the high requirements of the computation-intensive and delay-sensitive applications becomes the bottleneck for providing satisfactory user experience, and thereby may defer the advent of new business opportunities for prospective applications in future sixth generation (6G) wireless systems of the 2030s [2],[3].

To overcome this challenge, multi-access edge computing (MEC) has recently emerged as a new computing paradigm to provide cloud-computing capability within the radio access network (RAN) and received significant attention from both academia and industry [4]. The main idea of MEC is to shift the computing, memory, and storage resources to the edge of the RAN, e.g., 5G New Radio (NR) gNodeB, in close proximity to the users, e.g., SMDs in 5G scenario and connected things in IoT system. Correspondingly, the SMDs can offload the computation-intensive workload to physically proximate computing servers that hosted by a micro data center (MDC), e.g., cloudlets, which is integrated at the 5G NR gNodeB, instead of relying on a remote cloud. Thereby, the MEC has potentials to bring many benefits, such as reduced latency, enhanced security, improved energy saving, alleviated backhaul congestion, enhanced user experience, etc [5].

Despite its appealing benefits, a wide variety of mobile applications and wireless services for MEC system make network operators and service providers very difficult to optimize network management and service provision, to identify a specific application or service for each physical device running on their networks. To deal with this problem, network functions virtualization (NFV) has been recently recognized as a promising technique for MEC system, by decoupling network functions from the underlying dedicated hardware devices, which is also referred to as physical machines (PMs) [5],[6]. With the help of the virtualization capabilities offered by NFV, network functions can be implemented as the form of software, i.e., virtual machines (VMs), running on top of the sharing commoditized PMs by using virtualization software platform, e.g., the Hypervisor, at the virtualization layer, as shown in Fig. 1¹. Such a decoupling and sharing paradigm during the virtualization process not only enables the agility of network configuration by dynamically responding to new network services at acceptable maintenance costs, but also facilitates the opportunity to realize scalable, elastic, and programmable

¹ We would like to mention that VMs run their applications (APPs) according to the type of their respective operating system (OS) specified by the users, i.e., the SMDs, on top of the virtualization layer, as illustrated in Fig. 1.

next-generation wireless networks [7]. For the NFV-enabled MEC architecture, virtualized network functions (VNFs) in terms of virtual resources (e.g., computing, memory, and storage resources) can be instantiated at the network edge, e.g., the MDC integrated at the 5G NR gNodeB, which is described as virtualized MDC (vMDC) acting as a service provider for the virtualized edge computing. Particularly, as a virtualization infrastructure in virtualized MEC system, the resourceful vMDC is capable of creating multiple VMs that are simultaneously running on the underlying computation-enabled PMs in a flexible and cost-effective manner. With the virtualized MEC, the SMDs can purchase these virtualized resources in the form of customized VM instances as services offered by the vMDC for executing the offloaded computation workload.

From the perspective of service provider for edge computing, the costs of operation and maintenance primarily arise from the total energy consumption of dedicated hardware devices, i.e., PMs, hosted by the vMDC in the virtualized MEC [8]-[10]. To save the costs, a natural idea is to allow several VM instances to concurrently run on top of a single PM through the virtualization software platform. With the increasing scale of the virtualized MEC, there will be more energy consumption and more associated costs due to the impact of a large number of active PMs. Therefore, it is imperative to optimize the placement of plenty of VM instances over the underlying hardware devices, i.e., the efficient mapping of VM instances on top of the suitable PMs, aiming to increase the utilization of physical resources and reduce the total energy consumption. Under the scenario of the virtualized MEC, it is of paramount importance to provide solutions that effectively achieve the optimal VM placement on top of the PMs and accordingly assign the virtual resources to the SMDs while satisfying the requirement of the computation-intensive workload.

1.1 Related Work and Motivation

Currently, numerous research efforts have dealt with the problem of VM placement for data centers, and most of them focused on the scenario of cloud computing [8]-[14]. In [8], Liu *et al.* proposed an ant colony system based approach to achieve the VM placement by effectively minimizing the number of active physical servers to reduce the energy consumption for cloud computing. To capture the balance between saving PM power and guaranteeing VM performance, Zhao *et al.* in [9] developed a power-aware and performance-guaranteed VM placement scheme from the perspective of both cloud providers and users through an ant colony optimization method. In [10], Dai *et al.* presented two greedy approximation algorithms to intelligently place VMs on top of the underlying physical servers in data center to reduce the total energy consumption while satisfying the tenants' service level agreements. Based on the multi-linear programming, Rampersaud and Grosu in [11] proposed a shared perception online algorithm for VM packing. Particularly, cloud providers should provide a large number of users with the on-demand VM instances and package their associated vectors to minimize the number of physical servers aiming to reduce the total operational costs. In [12], Guo *et al.* designed a shadow routing based VM placement algorithm by enabling the VMs to share the CPU/memory resources on the same PM in large data centers within a network cloud. To improve the energy efficiency of cloud

data centers, Gharehpasha and Masdari in [13] combined the sine cosine algorithm and ant lion optimizer as discrete multi-objective and chaotic functions to obtain an optimal VM assignment by minimizing the power consumption for the number of active PMs. In [14], Wei *et al.* presented a joint bin-packing heuristic and genetic algorithm to balance the use of multiple physical resources in cloud data centers, aiming to reduce the fragmentation of resources and to maximize the service rate of VM placement. All the above works in [8]-[14], even though they provide valuable insights on the potential of efficient VM placement on top of the physical servers towards the overall performance improvement for cloud data centers, cannot be directly applied to the virtualized MEC scenario by integrating the NFV and the MEC framework.

On the other hand, the combination of the NFV paradigm and the MEC system to improve the overall performance of the virtualized MEC has attracted interest in the research community [15]-[19]. In [15], Li *et al.* proposed a polynomial time heuristic algorithm to solve the problem of the VNF placement for the request of service function chain (SFC) in the NFV-enabled edge computing. In [16], Tan *et al.* developed a framework of the full-duplex-based virtual small cell network to achieve both the MEC and the caching for two kinds of heterogeneous services, i.e., high-data-rate service and computationally sensitive service, aiming to save the cost of backhaul resources and to ensure the delay requirements of users. In [17], Liang *et al.* presented an offloading scheduling method to solve the problem of joint radio-and-computation resource allocation for the multi-user virtualized MEC systems with I/O interference, i.e., the interference between VMs sharing the same physical platform. The proposed method obtained the performance improvement of two system metrics, i.e., sum offloading throughput maximization and sum mobile energy consumption minimization. In [18], Chen *et al.* proposed a double deep Q-network based learning algorithm to solve the stochastic computation offloading problem in a sliced RAN with virtualized MEC system, with the objective of maximizing the long-term utility performance without knowing a priori knowledge of network dynamics. In [19], Rezvani *et al.* proposed a two-phase resource allocation framework for a parallel cooperative joint multi-bitrate video caching and transcoding in heterogeneous virtualized MEC networks. The aim of this developed framework is to jointly optimize both available physical resources with user association and request scheduling for maximizing the slice revenues. However, the above works only explored the application of NFV in the MEC environment from different perspectives, e.g., VNF placement of SFC [15] as well as joint computation offloading and multi-dimensional resource allocation [16]-[19], without investigating the VM placement problem over the underlying PMs under the virtualization settings.

Meanwhile, many recent works have been dedicated to the VM placement problem in the virtualized MEC scenario. In [5], Zhang *et al.* developed an optimization framework of energy consumption minimization for computing and offloading by jointly optimizing the VM placement matrix and the number of PMs via the non-uniform weighted hypergraph model. In [20], Wang *et al.* formulated a mathematical model to minimize the hardware consumption required by the VMs for supporting the

given workloads in the virtualized MEC scenario, while meeting the requirements of the heterogeneous latency for different computation-intensive applications. By capturing the impact of users' mobility on the dynamicity of the virtualized MEC system, Plachy *et al.* in [21] proposed an effective algorithm to achieve the flexible selection of communication path together with dynamic VM placement according to the expected users' movement. Zhao and Liu in [22] designed a latency aware heuristic placement algorithm to obtain the optimal placement of VM replica copies, aiming at the minimization of the average response time of deploying multiple applications among several MEC servers in the virtualized MEC networks.

The related research in [5] is heuristic, although the authors only investigated the effect of energy efficiency on the VM placement within the scenario of virtualized MEC. By contrast, we extensively consider a tradeoff to effectively balance the energy consumption and computing/offloading delay for VM placement. This is due to the fact that both the computation delay and energy consumption are critical to the users during computation offloading, especially for obtaining the optimal placement of VM instances on top of the PMs. Furthermore, the most prominent feature of the proposed VM placement schemes for whether cloud data centers [8]-[14] or the virtualized MEC [20]-[22] is that they assume that a single VM instance must be placed on top of one PM based on the conventional virtualization software platform, e.g., the Hypervisor. We wish to remark that the traditional Hypervisor cannot support the creation of a single VM instance that spans different PMs. However, it has been also revealed by the ScaleMP, Inc. that the vSMP ServerONE is capable of aggregating different PMs into a single high-performance VM². From practical point of view, one single VM instance can be technically placed across multiple different PMs under the virtualized MEC scenario.

Motivated by the above mentioned reasons, the work presented in this paper extends the traditional VM placement constraint in a more practical manner to realize the more complex mapping relation between VM instances and PMs in the vMDC of the virtualized MEC system. On the other hand, as already mentioned, the placement of the VM instances on top of the PMs should incorporate both the computation delay of the offloaded workload from the users and the energy consumption of the PMs. Unfortunately, none of these works have identified the tradeoff between energy consumption and computing/offloading delay for the VM placement over the underlying PMs. This research gap motivates us to pursue a solution for the problem of VM placement across multiple different PMs by minimizing the weighted sum based cost function with respect to the total energy consumption as well as the total computing/offloading delay.

1.2 Contributions and Organization

To the best of our knowledge, there have been no prior research works studying the performance balance between energy consumption and computing/offloading delay for the VM placement on top of the PMs in the virtualized MEC system. Particularly, the effect of the creation of a single VM instance

² Due to the space limitation, specific detail about the vSMP ServerONE is omitted here, and readers can refer to the website link (<https://www.scalemp.com/products/vsmp-serverone/>) for more detailed description.

spanning across multiple different PMs on the VM placement has yet not been thoroughly studied in the existing literature. For bridging the research gap, in this paper, we investigate the problem of VM placement by properly capturing the energy-delay tradeoff during computing and offloading in the virtualized MEC, aiming to minimize the weighted sum based cost function while satisfying the placement constraints. The main contributions of this paper can be summarized as follows:

- We develop a novel VM placement optimization framework to achieve the energy-delay tradeoff for local computing, computation offloading, and computing at the vMDC by taking into account the more complex mapping relation between the VM instances and the PMs. Our framework is the first time in the literature to identify a close coupling interplay between energy consumption and computing/offloading latency from the perspective of both the virtualization infrastructure and the users.
- We formulate the problem of VM placement as a pure 0-1 integer linear programming problem, which is NP-complete and hard to solve directly through the traditional centralized exhaustive method, especially with the increasing scale of the virtualized MEC system. To capture such a tradeoff, the objective of the VM placement is to minimize the weighted sum based cost function for computing and offloading by optimizing the VM instance placement relations on top of the PMs.
- To tackle this problem with a reduced time complexity, we transform this VM placement problem as a many-to-many two-sided matching problem between the VM instances and the PMs. For describing the specific utility obtained by a matching agent over others, we define the preference value paradigm of a given VM instance to a given PM by taking into account the placement constraints. Based on that, the stability of matching is further formulated to minimize the weighted sum based cost function for computing and offloading. Moreover, for obtaining a two-sided matching between the VM instances and the PMs, we propose an effective algorithm by using the SPA-(S, P) algorithm with low-complexity to identify the tradeoff between energy consumption and computing/offloading latency.

The rest of this paper is organized as follows. In Section 2, we introduce the system model and formulate the weighted sum based cost minimization problem to achieve the energy-delay tradeoff for the VM placement. In Section 3, we provide the many-to-many two-sided matching approach to solve this problem and further present an effective algorithm with low-complexity to derive the feasible solution. Simulation results are discussed in Section 4. Finally, Section 5 concludes this paper.

2. System Model and Problem Formulation

2.1. System Overview

As shown in Fig. 1, we consider an uplink NFV-enabled MEC system scenario, where one 5G NR gNodeB integrated with a vMDC serves K SMDs. The SMDs are randomly distributed within the coverage area of the gNodeB. Denote $\mathcal{K} = \{1, 2, \dots, K\}$ as the set of all the SMDs each with a

computation workload of computation-intensive and delay-sensitive applications to be executed. We assume that each SMD has limited computation capability due to the energy- and size-constrained computing resources, which results in a large amount of consumed energy and time to complete the execution of workload. As a consequence, with the aid of the resourceful ν MDC in the virtualized MEC, every SMD adopts a partial computation offloading rule. In this way, the computation workload of the SMD can be processed locally or potentially offloaded to the ν MDC via the gNodeB using uplink transmission. We focus on one particular system time frame with duration T . Within duration T , we assume that the location of each SMD remains unchanged. For convenience, the size of computation data including the program codes and input parameters is adopted to characterize the computation workload for every SMD. We then define the computation workload of SMD k as W_k , which is described by $W_k = \{L_k, X_k\}$, where L_k (in bits) is the total data size of workload for SMD k , and X_k (in bits) is the data size of workload offloaded to the ν MDC, for $0 < X_k < L_k$ and $k \in \mathcal{K}$.

Fig.1 An illustration of a virtualized MEC scenario consisting of K SMDs and one 5G NR gNodeB equipped with a ν MDC.

With the partial computation offloading rule, every SMD requires a lot of physical resources, including computing, memory, storage, and network resources, to fulfill the application requirements for its computation-intensive workload³. We thus assume that the ν MDC owns M PMs that are used to provide the physical resources for executing the computation workload offloaded from all the SMDs within duration T . The set of all the PMs in the ν MDC is represented by $\mathcal{M} = \{1, 2, \dots, M\}$. For analytical simplicity, we only consider two types of physical resources for every PM, i.e., CPU kernels and memory sizes. Particularly, we define the available physical resources of PM m as an available resource vector $\mathbf{r}_m = (\alpha_m, \beta_m)$, where α_m is the number of available CPU kernels and β_m is the amount of available memory sizes in GB, for $m \in \mathcal{M}$. Let us further devise a resource requirement vector $\mathbf{I}_m = (u_m, v_m)$, where u_m and v_m refer to the number of required CPU kernels and the amount of required memory sizes in GB, respectively, to maintain the normal operation of PM m when it is idle, for $m \in \mathcal{M}$.

³ In what follows, unless otherwise stated, we use the terms “computing resource” and “CPU kernels” interchangeably.

To implement the virtualized MEC paradigm, a great number of VMs should be created by the vMDC to achieve the simultaneous operations on top of the underlying PMs by using virtualization software platform, as shown in Fig.1. Meanwhile, the SMDs can purchase these customized VM instances offered by the vMDC for computation offloading. To this aim, it is further supposed that there are N VM instances which are concurrently running on top of M PMs in the vMDC within duration T . The set of all the VM instances is denoted as $\mathcal{N} = \{1, 2, \dots, N\}$. We also assume that each SMD can only purchase limited number of VM instances created by the vMDC, and the total number of VM instances purchased by all the SMDs should be equal to the size of VM instance set $\mathcal{N}$. Specifically, let δ_k stand for the number of VM instances purchased by SMD k , for $\delta_k \ll N$. Denote by $\Delta_k = \{1, 2, \dots, \delta_k\}$ as the set of δ_k VM instances purchased by SMD k , for $\Delta_k \subset \mathcal{N}$. From the perspective of K SMDs, we can obviously have:

$$\Delta_1 \cap \Delta_2 \cap \dots \cap \Delta_k \dots \cap \Delta_K = \emptyset, \quad (1)$$

$$\Delta_1 \cup \Delta_2 \cup \dots \cup \Delta_k \dots \cup \Delta_K = \mathcal{N}. \quad (2)$$

Let us then characterize VM instance n as a virtual resource vector $\mathbf{\Gamma}_n = (c_n, s_n)$, where c_n and s_n are the number of CPU kernels and the amount of memory sizes in GB, respectively, for $n \in \mathcal{N}$. We further use $\alpha_{n,m}$ and $\beta_{n,m}$ to stand for the number of CPU kernels and the amount of memory sizes of VM instance n that are placed on top of PM m at the same time, respectively. In order to represent the placement relationship between VM instance n and PM m , we then introduce an integer binary variable $b_{n,m}$, which can be determined by:

$$b_{n,m} = \begin{cases} 1, & \text{if VM instance } n \text{ is placed on PM } m, \forall n \in \mathcal{N}, m \in \mathcal{M}, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

Thereby, elements c_n and s_n in virtual resource vector $\mathbf{\Gamma}_n$ of VM instance n can be respectively interpreted as:

$$\sum_{m=1}^M b_{n,m} \alpha_{n,m} \leq c_n, \quad \forall n \in \mathcal{N}, \quad (4)$$

$$\sum_{m=1}^M b_{n,m} \beta_{n,m} \leq s_n, \quad \forall n \in \mathcal{N}. \quad (5)$$

2.2. Communication Model

As previously mentioned, the partial computation offloading rule is employed for every SMD due to the powerful capabilities for computing, storage, and network resources of the vMDC integrated at the gNodeB for the virtualized MEC. With such an offloading rule, the considered time frame with duration T to execute the whole computation workload for K SMDs can be divided into three stages, i.e., offloading stage, computing stage, and downloading stage, as depicted in Fig 2. In order for K SMDs to offload their workload to the gNodeB for computing via uplink transmissions, it will be assumed that computation offloading for K SMDs in the offloading stage is operated on the same channel with

bandwidth denoted by B . To avoid the co-channel interference, similar to [23],[24], we adopt a TDMA-based co-channel access protocol to efficiently schedule the multiple access of K SMDs for computation offloading. As such, in the offloading stage, K SMDs offload their computation workload to the gNodeB one by one within every allocated time slot, as shown in Fig. 2. Let t_k^o be the time slot allocated to SMD k for offloading its workload to the gNodeB within duration T , for $k \in \mathcal{K}$. In the computing stage, the vMDC at the gNodeB performs the edge execution of the offloaded computation workload for K SMDs within the allocated time slot. We denote T^c as the time slot allocated to the vMDC for completing the execution of workload from K SMDs. Similar to [23],[25], we can neglect the time consumed at the downloading stage for sending the computation results back to K SMDs due to higher transmit power of the gNodeB and smaller number of bits associated with the computation results.

Fig. 2 The time slot allocation of the system time frame with duration T .

Based on the time slot allocation as already introduced, we thus only consider the problem of uplink transmission for computation offloading in the offloading stage where each SMD offloads its workload to the gNodeB. For analytical simplicity, we assume that uplink transmission follows the quasi-static block fading channel model, in which the channel gain remains unchanged within the considered time slot, but varies independently from one to another. The channel gain between SMD k and the gNodeB is denoted by $\eta_k = g_k / d_k^{0.5\theta}$, where g_k is the small-scale fading channel gain between SMD k and the gNodeB, d_k is the distance between SMD k and the gNodeB, and $\theta \geq 2$ is the path loss exponent. As in [26], the small-scale fading channel gain is subject to the Rayleigh distribution, i.e., $g_k \sim \mathcal{CN}(0,1)$ with $\mathcal{CN}(\nu_1, \nu_2)$ is the complexed Gaussian distribution with mean ν_1 and variance ν_2 . Let p_k denote the transmit power of SMD k when it offloads its workload to the gNodeB via uplink transmission, with $P_k^{\max}$ being its maximum allowable value, for $p_k \leq P_k^{\max}$. Particularly, the maximum transmit power $P_k^{\max}$ (in Watt) of SMD k for uplink transmission can be calculated by:

$$P_k^{\max} = 10^{0.1P_0(\text{dBm}) - \theta \lg \frac{d_k}{d_0} - 3}, \quad (6)$$

where P_0 is the receiving reference power by the gNodeB at a reference distance d_0 . Depending on the TDMA-based co-channel access protocol, the achievable rate (in bps) for uplink transmission from SMD k to the gNodeB can be expressed as:

$$R_k = B \log_2 \left(1 + \frac{p_k \eta_k}{\sigma^2} \right), \quad (7)$$

where σ^2 is the power of additive white Gaussian noise at the gNodeB. To simplify the analysis, we suppose that every SMD has a fixed transmit power for uplink transmission, with an assumption that it is proportional to its maximum transmit power during the offloading phase. Thus, the transmit power of SMD k can be specified by $p_k = \varepsilon P_k^{\max}$, where $\varepsilon \in (0, 1]$ refers to an adjustment factor.

2.3. Computation Model

3.3.1. Local Computing

Since only X_k bits in computation workload $W_k = \{L_k, X_k\}$ are offloaded to the vMDC integrated at the gNodeB via uplink transmission, the data size of the remaining workload that is required to be computed locally at SMD k can be denoted by $L_k - X_k$, for $k \in \mathcal{K}$. We assume that each SMD has a given number of CPU kernels as its local computing resources, and each CPU kernel has fixed and equal computing capability, which is measured by CPU cycles per second. Let μ_k and f_k represent the number of CPU kernels owned by SMD k and the local computation capability of each CPU kernel for SMD k , respectively. We also denote by $\wp_1$ as the number of CPU cycles required to execute one bit of raw data for local computing at the SMDs, which is determined by the feature of computation-intensive and delay-sensitive applications. Note that the number of CPU cycles required to complete one bit of raw data is assumed to be equal for every SMD. As such, the total number of required CPU cycles for SMD k to accomplish the data size of $L_k - X_k$ bits locally can be written by $\wp_1 (L_k - X_k)$. Consequently, the computation delay of processing the computation workload W_k locally can be calculated as:

$$t_k^L = \frac{\wp_1 (L_k - X_k)}{\mu_k f_k}. \quad (8)$$

According to [27], we model the energy consumption of per CPU cycle at SMD k as $\zeta_k (\mu_k f_k)^2$, where ζ_k is an effective capacitance coefficient that depends on the chip architecture of CPU at SMD k . Correspondingly, the energy consumption for local computing at SMD k can be obtained by:

$$E_k^L = \wp_1 \zeta_k (\mu_k f_k)^2 (L_k - X_k). \quad (9)$$

3.3.2. Computation Offloading

By adopting the TDMA-based co-channel access protocol to coordinate the access of multiple SMDs for computation offloading in the virtualized MEC, every SMD then offloads its computation workload to the gNodeB based on the allocated time slot, as shown in Fig. 2. With X_k bits workload offloaded to the vMDC, the total amount of uplink transmitted data for SMD k can be actually characterized by τX_k , where τ is the communication overhead composed of forward error control coding and data encryption during uplink transmission. Thus, based on the achievable rate R_k for uplink transmission, the offloading delay of SMD k within the allocated time slot can be modeled as:

$$t_k^O = \frac{\tau X_k}{R_k} = \frac{\tau X_k}{B \log_2 \left(1 + \frac{p_k \eta_k}{\sigma^2} \right)}. \quad (10)$$

Accordingly, the energy consumption of computation offloading for SMD k is given by:

$$E_k^O = t_k^O p_k = \frac{\varepsilon \tau X_k P_k^{\max}}{B \log_2 \left(1 + \frac{\varepsilon \eta_k P_k^{\max}}{\sigma^2} \right)}. \quad (11)$$

3.3.3. Computing at the vMDC

For computing at the vMDC, the VM instances purchased by the SMDs execute the total amount of received data offloaded from the SMDs under the scenario of the virtualized MEC. Let us employ ρ to denote the average computing capability of one CPU kernel for every VM instance, which is also measured by CPU cycles per second. Recall that VM instance n owns c_n CPU kernels under the virtualized MEC framework as already stated. Thus, the computing capability of VM instance n is specified by ρc_n , for $n \in \mathcal{N}$. Then the computing capability of δ_k VM instances purchased by SMD k is equal to $\sum_{n \in \Delta_k} \rho c_n$. Let $\wp_2$ be the number of CPU cycles required to execute one bit of raw data for computing at the vMDC. Thereby, from the perspective of the vMDC, the computation delay of the workload offloaded from SMD k can be expressed as:

$$t_k^C = \frac{\wp_2 \tau X_k}{\sum_{n \in \Delta_k} \rho c_n} = \frac{\wp_2 \tau X_k}{\sum_{n \in \Delta_k} \sum_{m=1}^M \rho b_{n,m} \alpha_{n,m}}. \quad (12)$$

As a result, the computation delay of the overall workload from K SMDs at the vMDC can be written by:

$$T^C = \sum_{k=1}^K t_k^C = \sum_{k=1}^K \sum_{n \in \Delta_k} \sum_{m=1}^M \frac{\wp_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}}. \quad (13)$$

We focus on the energy consumption in the computing stage in terms of the number of active PMs. According to [27], as for the underlying PM, both the energy consumption and the CPU utilization is approximately linear. Essentially, the energy consumption of the PM consists of static power consumption and dynamic power consumption. When the PM is idle, the static power consumption is related to the PM's hardware configuration. Meanwhile, the dynamic power consumption of the PM is associated with the CPU utilization rate. To simplify the analysis of the problem, we define the ratio of the number of CPU kernels already assigned to the total number of CPU kernels for PM m as the CPU utilization rate. More specifically, the CPU utilization rate of PM m can be calculated by:

$$\gamma_m = \frac{\sum_{n=1}^N b_{n,m} \alpha_{n,m}}{\alpha_m + u_m}. \quad (14)$$

Therefore, as in [27], by adopting its CPU utilization rate, the power consumption of PM m can be defined as:

$$\Xi_m = \Xi_m^{\text{idle}} + (\Xi_m^{\text{max}} - \Xi_m^{\text{idle}}) \gamma_m, \quad (15)$$

where Ξ_m^{idle} is the static power consumption when PM m is idle and Ξ_m^{max} is the maximum power consumption when the PM m is fully loaded. Note that the static power consumption of the PM is almost 30% of its maximum power consumption, and the maximum power consumption of the PM is approximately 75% of the nameplate value [27]. Hence, the energy consumption of M PMs during the computing stage can be given by:

$$\begin{aligned} E^C &= T^C \sum_{m=1}^M \Xi_m \\ &= T^C \sum_{m=1}^M (\Xi_m^{\text{idle}} + (\Xi_m^{\text{max}} - \Xi_m^{\text{idle}}) \gamma_m) \\ &= \sum_{k=1}^K \sum_{n \in \Delta_k} \sum_{m=1}^M \frac{\wp_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}} \left(\sum_{m=1}^M E_m^{\text{idle}} + \sum_{m=1}^M \sum_{n=1}^N \frac{b_{n,m} \alpha_{n,m} (\Xi_m^{\text{max}} - \Xi_m^{\text{idle}})}{\alpha_m + u_m} \right). \end{aligned} \quad (16)$$

2.4. Problem Formulation

By extending computation and storage resources to the network edge in close proximity to end users, the virtualized MEC contributes to more energy saving for the SMDs compared with pure local computing, but resulting in the increased offloading delay of the SMDs. Moreover, the placement of the VM instances on top of the PMs has a significant impact on not only the computation delay of the offloaded workload from the SMDs, but also the energy consumption of the PMs in the vMDC integrated at the gNodeB. Keeping this fact in mind, both the computing delay and energy consumption are critical to the SMDs during computation offloading while finding the optimal placement of VM instances on top of the underlying PMs. As a result, there is a tradeoff between energy consumption and computing/offloading delay due to their coupling interplay under the considered virtualized MEC framework. Our objective is thus to identify the tradeoff between energy consumption and computing/offloading delay from the perspective of the vMDC and all the SMDs by optimizing the placement relations between the VM instances and the PMs. To capture such a tradeoff, we employ a weighted sum based cost function, which is calculated by the weighted sum of energy consumption cost and computing/offloading delay cost. More precisely, we define the weighted sum based cost function as follows:

$$F(b_{n,m}) = w E_{\text{total}} + (1 - w) \chi_T T_{\text{total}}, \quad (17)$$

where $w \in [0, 1]$ is the weighting factor that reflects the tradeoff between energy consumption and computing/offloading delay, χ_T is the normalization factor used to ensure the same range for two devised costs, E_{total} is the total energy consumption cost, and T_{total} is the total computing/offloading delay cost. Since the energy consumption for every SMD including the local computing and computation offloading as well as for active PMs at the vMDC have been obtained through (9), (11), and (16), the total energy consumption cost is given by:

$$\begin{aligned}
E_{\text{total}} &= \sum_{k=1}^K (E_k^L + E_k^O) + E^C \\
&= \sum_{k=1}^K \left(\wp_1 \zeta_k (\mu_k f_k)^2 (L_k - X_k) + \frac{\varepsilon \tau X_k P_k^{\max}}{R_k} \right) \\
&\quad + \sum_{k=1}^K \sum_{n \in \Delta_k} \sum_{m=1}^M \frac{\wp_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}} \left(\sum_{m=1}^M E_m^{\text{idle}} + \sum_{m=1}^M \sum_{n=1}^N \frac{b_{n,m} \alpha_{n,m} (\Xi_m^{\max} - \Xi_m^{\text{idle}})}{\alpha_m + u_m} \right).
\end{aligned} \tag{18}$$

Meanwhile, based on the local computing delay, computation offloading delay, and computing delay at the vMDC for every SMD by using (8), (10), and (12), the total computing/offloading delay cost can be expressed by:

$$\begin{aligned}
T_{\text{total}} &= \sum_{k=1}^K (t_k^L + t_k^O + t_k^C) \\
&= \sum_{k=1}^K \left(\frac{\wp_1 (L_k - X_k)}{\mu_k f_k} + \frac{\tau X_k}{R_k} + \sum_{n \in \Delta_k} \sum_{m=1}^M \frac{\wp_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}} \right).
\end{aligned} \tag{19}$$

Under the above setup, we aim to minimize the weighted sum based cost function for computing and offloading within duration T by optimizing the VM instance placement relations on top of the PMs. The optimization problem can be mathematically formulated as:

$$\min_{b_{n,m}} F(b_{n,m}) = w E_{\text{total}} + (1 - w) \chi_T T_{\text{total}} \tag{20a}$$

$$\text{s.t. } \sum_{m=1}^M b_{n,m} \alpha_{n,m} \leq c_n, \quad \forall n, \tag{20b}$$

$$\sum_{m=1}^M b_{n,m} \beta_{n,m} \leq s_n, \quad \forall n, \tag{20c}$$

$$b_{n,m} \in \{0, 1\}, \quad \forall n, m. \tag{20d}$$

In (20), constraints (20b) and (20c) represent the placement constraints of VM instances on top of the PMs, i.e., the number of CPU kernels and the amount of memory sizes of the VM instances. Constraint (20d) corresponds to the constraint of the binary placement variable, which also dictates whether VM instance is placed on top of the associated PM.

3. Proposed Two-Sided Matching Approach

In this section, we consider the solution to the optimization problem in (20) by identifying an optimal VM placement scheme. Note that the formulated problem in (20) is a pure 0-1 integer linear programming problem due to the binary variable for VM instance placement in objective function (20a) as well as two linear constraints in (20b) and (20c). In general, such a problem is NP-complete and computationally intractable. Especially, with the increasing scale of the virtualized MEC in which N and M become very large, it is extremely complex to solve the problem directly by adopting the traditional centralized exhaustive method. However, since the optimization problem in (20) contains only one binary variable, it can be thus converted into a matching problem [28]. Inspired by [29] and

[30], in what follows, we will propose the effective algorithm based on two-sided matching theory [28]-[31] for the sub-optimal solution of the formulated problem.

As noticed previously, such a binary variable reflects the placement of one given VM instance on top of one given PM, i.e., one-to-one mapping relation constraint, under the virtualized MEC scenario. In this way, we wish to remark that this work considers the more complex mapping relation between the VM instances and the PMs by capturing the constraint that one VM can be technically placed across multiple different PMs. Therefore, the many-to-many two-sided matching between the VM instances and the PMs is then invoked for obtaining a sub-optimal solution to the formulated problem in (20).

3.1 Preference and Preference List

The primary goal of the two-sided matching is to optimally match two sets of matching agents together, given their individual utilities. As for the classical stable marriage matching model, a man or a woman is called to be a matching agent. For our work, the matching agent refers to the VM instance or the underlying PM. As in [29], the preference or the preference value of an agent over other agents can be characterized by the specific utility value that provides the performance metric of each agent with respect to other agents. To obtain the matching result of the considered VM placement problem, it is of paramount importance to characterize the preference values for both the VM instances and the PMs. For ease of exposition, we give the definition of the preference value of a given VM instance to a given PM as follows.

Definition 1 (Preference Value): The degree of matching between the remaining resources of PM m and the resources required by VM instance n is defined as the preference value of VM instance n to PM m , for $n \in \mathcal{N}$ and $m \in \mathcal{M}$.

Based on Definition 1, we use $\Psi_{n,m}$ and $Q_{n,m}$ to denote the preference values of the number of CPU kernels and the amount of memory sizes belonging to VM instance n that is placed on top of PM m , respectively. Thereby, the preference values $\Psi_{n,m}$ and $Q_{n,m}$ can be respectively expressed by:

$$\Psi_{n,m} = \frac{\alpha_m - \sum_{n=1}^N b_{(n-1),(m-1)} \alpha_{(n-1),(m-1)} - \alpha_{n,m}}{\alpha_m + u_m}, \quad (21)$$

$$Q_{n,m} = \frac{\beta_m - \sum_{n=1}^N b_{(n-1),(m-1)} \beta_{(n-1),(m-1)} - \beta_{n,m}}{\beta_m + v_m}, \quad (22)$$

where $\alpha_m + u_m$ and $\beta_m + v_m$ are the total number of CPU kernels of PM m and the amount of memory sizes of PM m , respectively. From (21) and (22), if both the preference values $\Psi_{n,m}$ and $Q_{n,m}$ are less than zero, i.e., negative values, we note that PM m does not have enough physical resources to deploy VM instance n . Additionally, if the preference values of PM m with all the VM instances are negative, we further note that there will be no VM instance that can satisfy the deployment requirements of PM m .

Therefore, a new adopted PM is required to be turned on in an active mode for hosting the associated VM instances.

Accordingly, the preference matrix, denoted by $\mathbf{Z}_m$, between PM m and N VM instances can be specified by:

$$\mathbf{Z}_m = \begin{bmatrix} \Psi_{1,m} & Q_{1,m} \\ \Psi_{2,m} & Q_{2,m} \\ \vdots & \vdots \\ \Psi_{n,m} & Q_{n,m} \\ \vdots & \vdots \\ \Psi_{N,m} & Q_{N,m} \end{bmatrix}, \quad (23)$$

where column $[\Psi_{1,m}, \Psi_{2,m}, \dots, \Psi_{n,m}, \dots, \Psi_{N,m}]$ are the preference values of the number of CPU kernels of VM instance n that is placed on top of PM m , and column $[Q_{1,m}, Q_{2,m}, \dots, Q_{n,m}, \dots, Q_{N,m}]$ are the preference values of the amount of memory sizes of VM instance n that is placed on top of PM m . Since we mainly consider both the number of CPU kernels and the amount of memory sizes as two types of physical resources for every underlying PM, we thus employ q_1 and q_2 to represent the weights of CPU kernels and memory sizes, where $q_1 + q_2 = 1$, for $0 \leq q_1 \leq 1$, $0 \leq q_2 \leq 1$. As a result, the preference vector ω_m of PM m and N VM instances can be obtained as follows:

$$\begin{aligned} \omega_m &= \begin{bmatrix} \Psi_{1,m} & Q_{1,m} \\ \Psi_{2,m} & Q_{2,m} \\ \vdots & \vdots \\ \Psi_{n,m} & Q_{n,m} \\ \vdots & \vdots \\ \Psi_{N,m} & Q_{N,m} \end{bmatrix} \cdot [q_1 \quad q_2]^T \\ &= [\omega_{1,m} \quad \omega_{2,m} \quad \dots \quad \omega_{n,m} \quad \dots \quad \omega_{N,m}]^T, \end{aligned} \quad (24)$$

where $\omega_{n,m}$ represents the preference degree of VM instance n that is placed on top of PM m , and notation $[\cdot]^T$ stands for the transpose of a matrix. From (24), we wish to remark that the smaller the preference degree $\omega_{n,m}$, the more suitable for VM instance n to be placed on top of PM m . In order to enable both the PMs and the VM instances to obtain the more satisfactory matching objective, we then define the stability paradigm for such kind of matching, which is beneficial to both the underlying PMs and the VM instances. In particular, the formal definition of the stability of matching is provided in Definition 2 as follows.

Definition 2 (Stability): A matching $\mathcal{G}$ is said be stable, if there is no blocking pair (BP). A pair (n, m) is defined as a BP, for $n \in \mathcal{N}$ and $m \in \mathcal{M}$, if and only if all of the following conditions are satisfied:

- (1) n finds m acceptable;
- (2) either n is unmatched in $\mathcal{G}$, or n prefers m to $\mathcal{G}(n)$;

(3) either

(3.1) m is under subscribed and either of following three conditions is satisfied:

- (a) $\mathcal{G}(n) \in \mathcal{M}$, and n prefers m to $\mathcal{G}(n)$; or
- (b) $\mathcal{G}(n) \notin \mathcal{M}$, and n is under-subscribed; or
- (c) $\mathcal{G}(n) \notin \mathcal{M}$, and n is full and n prefers m to its current worst partner m_{worst} ;

(3.2) m is full and m prefers n to its current worst partner n_{worst} , and either of the following two conditions is satisfied:

- (a) $\mathcal{G}(m) \notin \mathcal{N}$;
- (b) $\mathcal{G}(m) \in \mathcal{N}$, and m prefers n to $\mathcal{G}(m)$;

We wish to remark that a stable matching is defined as a complete matching between a set of men and a set of women, wherein no BP exists in the classical stable marriage model [32]. In Definition 2, $\mathcal{G}(x)$ represents the partner/matching of player x in matching $\mathcal{G}$. To make it concise, we can easily have $\mathcal{G}(n) = m \in \mathcal{M}$. As such, the many-to-many matching $\mathcal{G}$ can be referred to be stable if matching $\mathcal{G}$ admits no BP. To find a stable matching, we need to firstly establish the preference list for the matching agents, i.e., the VM instances and the underlying PMs in the considered virtualized MEC scenario. It is noted that a preference list for each agent can be referred to as an ordered list based on the preferences over the other set of agents who he/she finds acceptable [32].

Based on the preference value paradigm as mentioned, let us utilize $\mathcal{F}$ and $\mathcal{S}$ to represent the preference lists for the VM instances and the underlying PMs, respectively. Particularly, we denote $\mathcal{F}_m$ as the preference list of PM m for N VM instances. Thus, from (24), preference list $\mathcal{F}_m$ of PM m for N VM instances ranks each VM instance in this acceptable set $\mathcal{W}_m = \{\omega_{1,m}, \omega_{2,m}, \dots, \omega_{n,m}, \dots, \omega_{N,m} \mid m \in \mathcal{M}\}$ in an ascending order according to the preferences $\mathcal{F}_m \in \mathcal{W}_m$. Hence, the preference lists of M PMs can be obtained as $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_m, \dots, \mathcal{F}_M\}$.

On the other hand, when selecting the VM instances to match its physical resources, we should consider the delay problem of the SMDs. Actually, the SMDs want to obtain the faster services and also need the lower delay for these applications. As a result, each SMD is more inclined to run on the underlying PM with the CPU kernel of higher computing capability measured by CPU cycles per second. As has been mentioned before, we employ ρ to represent the average computing capability of one CPU kernel for every VM instance. Actually, the CPU computing capability here is provided by each PM, which refers to its own CPU computing capability. For convenience, we assume that each CPU kernel of each PM has the same computation capability. Therefore, the CPU kernel's computing capability of M PMs can be formulated by a set $\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_m, \dots, \rho_M\}$. Meanwhile, the preference list of the VM instance n to M PMs is defined as the computation capability of one CPU kernel of the

PM, denoted by a set $\mathcal{S}_n$. Preference list $\mathcal{S}_n$ are arranged in descending order according to the CPU computing capability of PMs for each PM in this acceptable set $\mathcal{P} = \{\rho_1, \rho_2, \dots, \rho_m, \dots, \rho_M\}$, for $\mathcal{S}_n \in \mathcal{P}$. The preference lists of N VM instances is further denote by a set $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n, \dots, \mathcal{S}_N\}$.

3.2 Algorithm Design

Our objective is to obtain the performance balance between energy consumption and computing/offloading delay for the VM placement on top of the underlying PMs in the virtualized MEC system. It is worthy to mention that the PMs needs to match a more appropriate VM instance, while VM instances want to match the PMs with more CPU computing capability. In addition, a PM can host multiple VM instances, and a VM instance can be distributed across multiple different PMs, which is a many-to-many two-sided matching problem. To solve this matching problem and obtain the proper matching result, we refer to the problem of the classic student project allocation (SPA). Note that the SPA problem is a kind of problem that various students are shared to various projects (owned by different lecturers) under the aid of the lecturers. The project wants better students, and the students want to select the project they like. Due to the space limitation, specific detail about the SPA problem is omitted here, and readers can refer to [33] for more detailed description. Based on the SPA problem, we assume that the SMDs, VM instances and the PMs can be referred to as the lectures, the projects and the students, respectively. With the previous preference list as has been established, we can apply the SPA-(S, P) algorithm to find a two-sided matching between the VM instances and the PMs that provide the underlying physical resources. The details of the two-sided matching algorithm via the SPA are presented in Algorithm 1.

Algorithm 1 The Proposed Two-Sided Matching Algorithm Based on the SPA

Input: $\mathcal{N}$, $\mathcal{M}$, $\mathcal{F}$, $\mathcal{S}$;

Output: Matching $\mathcal{G}$;

Initialization: Set $\mathcal{G} = \emptyset$, and set all PMs free;

- 1: **while** some PM m is free and m has a non-empty preference list **do**
 - 2: **for all** $m \in \mathcal{M}$ **do**
 - 3: m proposes to the first entity n in $\mathcal{F}_m$, and remove n from $\mathcal{F}_m$;
 - 4: $\mathcal{G} \leftarrow \mathcal{G} \cup (n, m)$;
 - 5: **end for**
 - 6: **for all** $n \in \mathcal{N}$ **do**
 - 7: **while** n is over-subscribed **do**
 - 8: Find the worst pair $(n_{\text{worst}}, m_{\text{worst}})$ assigned to n in PMs' list;
 - 9: $\mathcal{G} \leftarrow \mathcal{G} / (n_{\text{worst}}, m_{\text{worst}})$;
 - 10: **end while**
 - 11: **end for**
 - 12: **end while**
 - 13: Terminate with matching $\mathcal{G}$.
-

We then analyze the computational complexity of the proposed two-sided matching algorithm. The computational complexity of Algorithm 1 mainly resides in the iteration for determination of the most appropriate VM instance from preference list as well as the total number of the PMs for all the iterations. At the first stage for the preferences of PM m to N VM instances, each iteration finds the most appropriate VM instance from preference list $\mathcal{F}_m$. For VM instance n that has been matched by PM m , we need to compare and select the most appropriate PM according to preference list $\mathcal{S}_n$ of VM instance n for all PMs. If VM instance n has been matched, we need to replace the worst PM that had been matched according to preference list $\mathcal{S}_n$. Hence, the complexity of this stage in Algorithm 1 can be derived as $O(m+n)$. At the second stage, the next PM loop will continue until M PMs are fully cycled in Algorithm 1. Obviously, the computational complexity of the second stage is determined by the order of $O(m)$. Therefore, the overall computational complexity of the proposed algorithm can be calculated as $O(m(m+n)) = O(m^2)$. It means that the complexity of the proposed two-sided matching algorithm polynomially increases with the number of PMs, which also indicates the scalability of the proposed algorithm in NFV-enabled MEC system scenario with the increasing number of the SMDs.

4. Simulation Results

In this section, we present the simulation results to verify our theoretical analysis and evaluate the performance of our proposed VM placement framework. We mainly study the impacts of the number of PMs and the number of VM instances on the performance of the proposed two-sided matching algorithm. Under the given simulation configuration, the performance of our proposed algorithm is demonstrated and discussed in terms of three optimization metrics, i.e., the total energy consumption, the total delay, and the total cost.

4.1 Simulation Settings

Regarding the simulations, we consider an uplink virtualized MEC system scenario, in which one 5G NR gNodeB integrated with a vMDC serves a given number of SMDs. Without loss of generality, we choose the number of the SMDs uniformly from the parameter range of [50,200] with the equal subinterval of 25. All the SMDs are assumed to be randomly distributed within the coverage area of the gNodeB in the simulations. We further assume that each SMD can purchase $\delta_k = 4$ VM instances from the resourceful vMDC for computation offloading.

Throughout the simulations, we choose the number of VM instances hosted by the vMDC from the parameter range of [200,800]. For convenience, we set the number of PMs according to the range of [50,300] with the equal subinterval of 50 to provide the underlying physical resources. Different from the traditional standards, we exploit the standard VM instance shapes provided by the Oracle cloud infrastructure for the settings of the specific resources for all the VM instances, i.e., CPU kernels and

memory size⁴. Particularly, we select a portion of the VM instance shapes from the Oracle for each VM instance. That is, the number of CPU kernels c_n and the amount of memory sizes s_n for each VM instance are randomly selected from the parameter range of [1,24] and the parameter range of [7,320]GB, respectively. For the PM's operation in the simulations, we assign the number of available CPU kernels α_m and the amount of available memory sizes β_m of each PM according to the range of [6,60] and the range of [480,2048]GB, respectively. Furthermore, the number of required CPU kernels u_m is set to [2,4,8], and the amount of required memory sizes v_m is always randomly allocated within the range of [60,480]GB, which are required by each PM to maintain its own normal operation. The other parameters used in the simulations unless stated otherwise are summarized in Table 1.

Table 1. Simulation parameters.

Parameter	Description	Value
P_0	Receiving reference power by the gNodeB	20dBm
d_0	Reference distance	50m
θ	The path loss exponent	2
g_k	Small-scale fading channel gain for SMD k	30
σ^2	Power of additive white Gaussian noise at the gNodeB	-1777dBm
B	Channel bandwidth	400 MHz
d_k	The distance from SMD k to the gNodeB	[58,59,60,65]m
ε	Adjustment factor	0.9
μ_k	Number of CPU kernels for SMD k	[4,6,8]
f_k	Local computation capability of CPU kernel for SMD k	5.35×10^8 cycle/s
τ	Communication overhead during uplink transmission	100 bits
$\wp_1$	CPU cycles of a bit at each SMD	10 cycles/bit
$\wp_2$	CPU cycles of a bit at the vMDC	0.5 cycles/bit
ζ_k	Effective capacitance coefficient of SMD k	10^{-26}
L_k	Total data size of computation workload for SMD k	1.37×10^8 bits
X_k	Data size of workload offloaded to the vMDC from SMD k	1.25×10^8 bits
χ_T	Normalization factor	1000

4.2 Results

Fig.2 shows the impact of the number of PMs on the total energy consumption under the given number of VM instances. As can be observed from this figure, the total energy consumption will markedly decrease with the increasing number of PMs from $M=50$ to $M=150$. Particularly, when the number of PMs is larger than 150, we can easily find that the total energy consumption tends to be a constant value for the given number of VM instances. This can be explained by the fact that the increase of the number of PMs will obviously enhance the overall computing capacity of the system, which results in the reduction of total energy consumption due to the improved utilization of the underlying dedicate physical resources via our proposed two-sided matching algorithm. As shown, the total energy consumption gradually tends to be the fixed constant value from 3.3878×10^4 to

⁴ Please refer to <https://cloud.oracle.com/compute/virtual-machine/features>, for the detail about the configuration of the CPU kernels and memory sizes.

1.6616×10^4 when $N=300$. Moreover, higher total energy consumption will be obtained with the growing number of VM instances. This phenomenon can be mainly explained by the fact that the computation tasks need to be assigned with the increasing number of VM instances. The results indicate that the choice of the numbers of the PMs M and the VM instances N has negligible effect on the total energy consumption of the virtualized MEC system.

Fig. 2 The impact of the number of PMs on the total energy consumption.

Fig. 3 The impact of the number of PMs on the total delay.

In Fig. 3, we compare the impact of the number of PMs on the total delay under the given number of VM instances. From Fig. 3, it is evident that the total delay will gradually decrease with the growing number of PMs from $M=50$ to $M=150$. It should be emphasized that the curve trends of the total delay become steady and have no changes while the number of PMs is greater than 150. The reason for this phenomenon is that the larger number of PMs generates much more computing capability for the

offloaded workload, which results in the reduction of the total delay, e.g., 11.9989 to 11.0333 while $N=400$. In addition, by adding the number of VM instances, we can observe that the total delay will be in the mode of the continued growth. Clearly, the larger computation workload will lead to the increase of the total delay, e.g., 5.5842 to 14.0755 when $M=100$. Therefore, it is confirmed that the proposed system performance can be improved by flexibly choosing the number of PMs based on the amount of offloaded computation tasks from SMDs to the vMDC.

Fig. 4 The impact of number of PMs on the total cost.

As depicted in Fig. 4, we demonstrate the comparison of the total cost according to the evolution of the number of PMs from $M=50$ to $M=300$ under the weighting factor w selected as 0.2, 0.5, and 0.8 and the given number of VM instances. It can be immediately observed that the total cost will obviously decrease as the number of PMs. Significantly, under the different settings of the weighting factor w , we can see that the total cost tends to be the fixed value when the number of PMs increases from $M=150$ to $M=300$. This is because that the higher number of PMs will lead to the faster calculation speed, resulting in the reduced energy consumption and the reduced delay from the perspective of the system-level. This result may further give rise to the lower total cost. That is, the total cost can decrease from

2.993×10^4 to 1.5582×10^4 when $N=400$ and $w=0.5$. Besides, it is worth noting that this observation emphasizes the importance of selecting the proper number of VM instances on the total cost. In addition, we can further find that the total cost under the smaller weighting factor is also obviously lower than that of the larger one. This result can be explained by the fact that the total cost by using our proposed optimization framework entirely depends on the proper selections of both the weighting factor and the number of VM instances.

Fig. 5 The impact of number of VMs on the total energy consumption.

As illustrated in Fig. 5, we look at the performance of the impact of the number of VM instances from $N=200$ to $N=800$ on the total energy consumption of system under the given number of the PMs. Regarding this figure, the total energy consumption based on the smaller number of VM instances outperforms the total energy consumption using the larger number of VM instances. In the meantime, the curves become more and more precipitous when the number of PMs decreases. Notice that the number of VM instances represents the number of computation tasks offloaded from the SMDs to the vMDC assigned to the PMs. Thus, this is because that our proposed two-sided matching algorithm can obtain the lower total energy consumption under the condition of the smaller number of VM instances, e.g., decreases from 8.8378×10^4 to 1.0273×10^4 when $M=100$. Beyond that, we can further see that the total energy consumption increases correspondingly with the decrease of the number of PMs. That is, the total energy consumption by using the number of PMs, $M=100$, is clearly more than that of the number of PMs, $M=250$. The explanation is that the more number of PMs may share the computational burden to reduce the associated energy consumption.

In Fig. 6, we further study the effect of the different numbers of VM instances from $N=200$ to $N=800$ on the effect of the total delay for our proposed VM placement framework. As shown in this figure, by increasing the number of VM instances, it is not difficult to find that the total delay is growing accordingly as expected. Apart from that, the curves of the total delay under different numbers of PMs

tend to be closest for each other. The major reason is the increase amount of the calculation workload assigned to the PMs by the VM instances will generate a lot of computational burden for the physical resources. This may further cause the higher total delay, e.g., significantly growth from 5.5842 to 22.3395 when $M=100$. However, although the total delay has been continuously growing, a shallow total delay can be relevantly actualized by adopting the larger number of PMs. This indicates that the more number of PMs is required to calculate computation tasks for reducing the total delay under the given number of VM instances.

Fig. 6 The impact of numbers of VMs on the total delay.

From the results of three sub-figures in Fig. 7, we further evaluate the performance of our proposed VM placement framework in terms of the total cost by increasing the number of VM instances from $N=200$ to $N=800$. It is clear from Fig. 7 that the total cost lies in a continuous growth when the number of VM instances is also growing, indicating that the low effectiveness of the considered system by employing the more number of VM instances. Particularly, when the weighting factor continuously magnifies, the curve has distinct variation, i.e., the improvement of the total cost is even more pronounced with a larger weighting factor. Moreover, it can be also observed that the total cost will be sustainably increasing with the gradual growth of the number of PMs for the three sub-figures corresponding to the different numbers of VM instances. This is due to the fact that the vMDC can own more physical resources to calculate the offloaded computation tasks, and thereupon then ascend the proposed system performance, i.e., the reduction of the total cost for our proposed framework. That is, the total cost varies from 7.9285×10^3 to 4.2922×10^4 using $w=0.5$ and $M=200$ when the number of VM instances increases from 200 to 800. This result implies that the proposed two-sided matching algorithm responds in a more cost-effective way to the increase of the number of the PMs by properly assigning the weighting factor in our formulated weighted sum based cost minimization problem

Fig. 7 The impact of numbers of VMs on the total cost.

Fig. 8 plots the impact of both the number of PMs and the number of VM instances on the total cost under the given normalization factor χ_T for the weighting factor $w=0.5$. In particular, the normalization factor χ_T is selected based on three constant values, i.e., 1000, 2000, and 3000. The result is obtained based on the number of PMs varying from $M=50$ to $M=300$ and the number of VM instances varying from $N=200$ to $N=700$. It is observed from this figure that the larger the normalization factor, the higher the total cost for our proposed VM placement framework under the constraint of the given number of PMs and the given number of VM instances. This is because that higher total delay with higher weight is more inclined to generate more and more total cost. From the figure, we can also see that the total cost will increase jointly with the decreasing number of PMs and the growing number of VM instances with the weighting factor $w=0.5$. This is due to the fact that the vMDC in the considered system can use more PMs to undertake the calculation of more offloaded computation tasks, thereupon reducing the total cost. However, if we increase the number of VM instances, there will be higher occupying physical resources, such that both the energy consumption and the delay will increase accordingly. Obviously, this result will give rise to more total cost. Particularly, the total cost will reduce from

1.5514×10^4 to 7.928×10^3 with the normalization factor $\chi_T=1000$ and the number of VM instances $N=200$. This infers that our proposed two-sided matching algorithm is able to offer the significant performance improvement by utilizing the proper normalization factor.

Fig. 8 The impact of both the number of PMs and the number of VMs on the total cost under different normalization factors with the weighting factor $w = 0.5$.

5. Conclusion

In this paper, we proposed a VM placement optimization framework for achieving a performance balance between energy consumption and computing/offloading delay in virtualized MEC system. This framework incorporated the more complex mapping relation between the VM instances and the PMs in comparison with the existing works. Motivated by that, we formulated a weighted sum based cost minimization problem as a pure 0-1 integer linear programming problem. Particularly, the formulated optimization problem was NP-complete and computationally intractable, especially for the increasing network scale. To tackle this problem with affordable computational complexity, we transformed this VM placement problem as a many-to-many two-sided matching problem. We further developed an effective algorithm using the SPA-(S, P) algorithm to obtain the sub-optimal solution of the formulated

problem. Finally, we validated and evaluated the performance of our proposed framework through numerical analysis by properly choosing both the number of PMs and the number of VM instances.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 61971032 and 61901115, the Natural Science Foundation of Hebei Province under Grants F2019402206, the Research Program for Top-notch Young Talents in Higher Education Institutions of Hebei Province, China under Grant BJ2017037, the Scientific Research Foundation of the Higher Education Institutions of Hebei Province of China under Grant QN2019059, the Natural Science Foundation of Shaanxi Province under Grant 2020JM-361, and the Fundamental Research Funds for the Central Universities under Grants FRF-TP-18-008A3 and 328201911.

References

- [1] L. Zhang, H. Zhao, S. Hou, Z. Zhao, H. Xu, X. Wu, Q. Wu, and R. Zhang, "A survey on 5G millimeter wave communications for UAV-assisted wireless networks," *IEEE Access*, vol. 7, pp. 117460–117504, July 2019.
- [2] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
- [3] E. C. Strinati, S. Barbarossa, J. L. Gonzalez-Jimenez, D. Ktenas, N. Cassiau, L. Maret, and C. Dehos, "6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 42–50, Sept. 2019.
- [4] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1409–1434, Third Quarter 2017.
- [5] L. Zhang, H. Zhang, L. Yu, H. Xu, L. Song, and Z. Han, "Virtual resource allocation for mobile edge computing: A hypergraph matching approach," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, Dec. 2019.
- [6] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212–262, Mar. 2018.
- [7] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, Mar. 2019.
- [8] X.-F. Liu, Z.-H. Zhan, J. D. Deng, Y. Li, T. Gu, and J. Zhang, "An energy efficient ant colony system for virtual machine placement in cloud computing," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 113–128, Feb. 2018.
- [9] H. Zhao, J. Wang, F. Liu, Q. Wang, W. Zhang, and Q. Zheng, "Power-aware and performance-guaranteed virtual machine placement in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1385–1400, Jun. 2018.
- [10] X. Dai, J. M. Wang, and B. Bensaou, "Energy-efficient virtual machines scheduling in multi-tenant data

- centers,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 210–221, Apr.-Jun. 2016.
- [11] S. Rampersaud and D. Grosu, “Sharing-aware online virtual machine packing in heterogeneous resource clouds,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2046–2059, Jul. 2017
- [12] Y. Guo, A. L. Stolyar, and A. Walid, “Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud,” *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 209–220, Jan.-Mar. 2018.
- [13] S. Gharehpasha and M. Masdari, “A discrete chaotic multi-objective SCA-ALO optimization algorithm for an optimal virtual machine placement in cloud data center,” *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–17, Nov. 2020.
- [14] W. Wei, K. Wang, K. Wang, H. Gu, and H. Shen. “Multi-resource balance optimization for virtual machine placement in cloud data centers,” *Computers & Electrical Engineering*, vol. 88, Dec. 2020.
- [15] D. Li, P. Hong, K. Xue, and J. Pei. “Virtual network function placement and resource optimization in NFV and edge computing enabled networks,” *Computer Networks*, vol. 152, pp. 12–24, 2019.
- [16] Z. Tan, R. Yu, X. Li, H. Ji, and V. C. M. Leung. “Virtual resource allocation for heterogeneous services in full duplex-enabled SCNs with mobile edge computing and caching,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 2, pp. 1794–1808, Feb. 2018.
- [17] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang. “Multiuser computation offloading and downloading for edge computing with virtualization,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4298–4311, Sept. 2019.
- [18] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis. “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
- [19] S. Rezvani, S. Parsaeefard, N. Mokari, M. R. Javan, and H. Yanikomeroglu. “Cooperative multi-bitrate video caching and transcoding in multicarrier NOMA-assisted heterogeneous virtualized MEC networks,” *IEEE Access*, vol. 7, pp. 93511–93536, Jul. 2019.
- [20] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang, and B. Mukerjee. “Virtual machine placement and workload assignment for mobile edge computing,” in *Proc. IEEE International Conference on Cloud Networking (CloudNet)*, Prague, Czech Republic, Sept. 2017.
- [21] J. Plachy, Z. Becvar, and E. C. Strinati. “Dynamic resource allocation exploiting mobility prediction in mobile edge computing,” in *Proc. IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Valencia, Spain, Sept. 2016.
- [22] L. Zhao and J. Liu, “Optimal placement of virtual machines for supporting multiple applications in mobile edge networks,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 6533–6545, Jul. 2018.
- [23] L. Zhang, Z. Zhao, Q. Wu, H. Zhao, H. Xu, and X. Wu, “Energy-aware dynamic resource allocation in UAV assisted mobile edge computing over social internet of vehicles,” *IEEE Access*, vol. 6, pp. 56700–56715, Oct. 2018.
- [24] F. Wang, J. Xu, X. Wang, and S. Cui, “Joint offloading and computing optimization in wireless powered mobile-edge computing systems,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

- [25] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.
- [26] Z. Yang, Z. Ding, P. Fan, and G. K. Karagiannidis, "On the performance of non-orthogonal multiple access systems with partial channel information," *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 654–667, Feb. 2018.
- [27] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proc. ACM International Symposium on Computer Architecture (ISCA)*, San Diego, CA, USA, Jun. 2007, pp. 13–23.
- [28] D. F. Manlove, *Algorithmics of Matching Under Preferences*. World Scientific, 2013.
- [29] S. Bayat, Y. Li, L. Song, and Z. Han, "Matching theory: Applications in wireless communications," *IEEE Signal Processing Magazine*, vol. 33, no. 6, pp. 103–122, Nov. 2016.
- [30] Y. Gu, W. Saad, M. Bennis, M. Debbah, Z. Han, "Matching theory for future wireless networks: Fundamentals and applications," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 52–59, May. 2015.
- [31] L. Gao, L. Duan, J. Huang, "Two-sided matching based cooperative spectrum sharing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 2, pp. 538–551, Feb. 2017.
- [32] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, Jan. 1962.
- [33] A. H. A. El-Atta and M. I. Moussa, "Student project allocation with preference lists over (student, project) pairs," in *Proc. International Conference on Computer and Electrical Engineering*, Dubai, United Arab Emirates, Dec. 2009.