Energy-delay tradeoff for virtual machine placement in virtualized multi-access edge computing: a two-sided matching approach

Long Zhang1 · Shanshan Zhuge1 · Yao Wang1 · Haitao Xu2 · Enchang Sun3,4

Received: 19 January 2021 / Accepted: 23 September 2021 / Published online: 6 October 2021
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2021

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 the 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 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 then 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 matching problem. Simulation results are presented to show the effectiveness of our proposed algorithm, and the normalization factor is of great significance to obtain the lower total cost.

Keywords Multi-access edge computing · Network functions virtualization · Virtual machine placement · Physical machine · Energy-delay tradeoff · Matching theory

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. (Zhang et al. 2019b, 2020b; Chettri and Bera 2020). 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 SMDs and the high requirements of 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 (Mao et al. 2016; Strinati et al. 2019; Zhang et al. 2020a).

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 (Li et al. 2021; Sun and Nakhai 2020; Filippou et al. 2020). 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. (Zhang et al. 2019a).

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 are also referred to as physical machines (PMs) (Zhang et al. 2019a; Yi et al. 2018; Yang et al. 2021). 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 the virtualization software platform, e.g., the Hypervisor, at the virtualization layer, as shown in Fig. 1.¹ 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 next-generation wireless networks (Hawilo et al. 2019; Kulkarni et al. 2020). 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 (Liu et al. 2018; Zhao et al. 2018; Dai et al. 2016). To save the costs, a natural idea is to allow several VM instances to concurrently run on top of a single PM using 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 for the system level. Under the scenario of the virtualized MEC, it is of paramount importance

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.
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 computation-intensive workload.

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 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.

Anticipating the great challenges in directly solving VM placement problem, we propose a two-sided matching approach for it as outlined in Fig. 2. Our approach converts the weighted sum based cost minimization problem (see Problem Domain in Fig. 2) for VM placement in the virtualized MEC scenario (see Scenario Domain in Fig. 2) into a many-to-many two-sided matching problem between the VM instances and the PMs. Additionally, to find a two-sided matching between the VM instances and the PMs, we develop an effective algorithm based on the classic student project allocation (SPA) algorithm with low-complexity (see Algorithm Domain in Fig. 2). 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 completely close coupling interplay between energy consumption and computing/offloading latency form 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. We first introduce the related work in Sect. 2. Section 3 describes the system model, followed by a formulation of the optimization problem. In Sect. 4, we provide the 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 Sect. 5. Finally, Sect. 6 concludes this paper.

2 Related work

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 (Liu et al. 2018; Zhao et al. 2018; Dai et al. 2016; Rampersaud and Grosu 2017; Guo et al. 2018; Gharehpasha and Masdari 2020; Wei et al. 2020). In Liu et al. (2018), an ant colony based approach was proposed 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, a power-aware and performance-guaranteed VM placement scheme was developed from the perspective of both cloud providers and users through an ant colony optimization method in Zhao et al. (2018). In Dai et al. (2016), two greedy approximation algorithms were presented 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, in Rampersaud and Grosu (2017), a shared perception online algorithm was proposed 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 Guo et al. (2018), a shadow routing based VM placement algorithm was designed by enabling the VMs to share the CPU/memory resources on the same PM in large data centers within the network cloud scenario. To improve the energy efficiency of cloud data centers, Gharehpasha and Masdari (2020) combined the sine cosine algorithm and ant lion optimizer as the discrete multi-objective and chaotic functions to obtain an optimal VM assignment by minimizing the power consumption for the number of active PMs. In Wei et al. (2020), a joint bin-packing heuristic and genetic algorithm was presented 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 (Liu et al. 2018; Zhao et al. 2018; Dai et al. 2016; Rampersaud and Grosu 2017; Guo et al. 2018; Gharehpasha and Masdari 2020; Wei et al. 2020), 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 (Li et al. 2019; Tan et al. 2018; Liang et al. 2019; Chen et al. 2019; Rezvani et al. 2019). In Li et al. (2019), a polynomial time heuristic algorithm was proposed to solve the problem of the VNF placement for the request of service function chain (SFC) in the NFV-enabled edge computing. In Tan et al. (2018), a framework of the full-duplex-based virtual small cell network was developed to achieve both the MEC and the caching optimization 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 Liang et al. (2019), an offloading scheduling method was presented to solve the problem of joint radio-and-computation resource allocation for the multi-user virtualized MEC systems with the 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, namely, the sum offloading throughput maximization as well we the sum mobile energy consumption minimization. In Chen et al. (2019), a double deep Q-network based learning algorithm was proposed to solve the stochastic computation offloading problem in the 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 Rezvani et al. (2019), a two-phase resource allocation framework was proposed for the parallel cooperative joint multi-bitrate video caching and transcoding in the 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 (Li et al. 2019) as well as joint computation offloading and multi-dimensional resource allocation (Tan et al. 2018; Liang et al. 2019; Chen et al. 2019; Rezvani et al. 2019), 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 sce- nario. In Zhang et al. (2019a), an optimization framework of energy consumption minimization was developed for computing and offloading by jointly optimizing the VM placement matrix and the number of PMs via the non-uniform weighted hypergraph model. In Wang et al. (2017), a mathematical model was formulated 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, an effective algorithm was proposed to achieve the flexible selection of communication path together with dynamic VM placement according to the expected users’ movement in Plachy et al. (2016). In Zhao and Liu (2018), a latency aware heuristic placement algorithm was designed 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 (Zhang et al. 2019a) 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 (Liu et al. 2018; Zhao et al. 2018; Dai et al. 2016; Rampersaud and Grosu 2017; Guo et al. 2018; Gharehplasha and Masdari 2020; Wei et al. 2020) or the virtualized MEC (Wang et al. 2017; Plachy et al. 2016; Zhao and Liu 2018) 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 also been revealed by the ScaleMP, Inc. that the innovative versatile ScaleMP (vSMP) Serve-rONE is capable of aggregating different underlying PMs into a single high-performance VM.² 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 completely 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.

3 System model and problem formulation

In this section, we introduce the system model for the VM placement in virtualized MEC system. We first provide an overview of the system scenario, and then, present the details on the communication model, followed by the computation model via the partial computation offloading rule. Based on the system model, we further discuss the problem formulation for energy-delay tradeoff for the VM placement in detail. The key notations and variables used throughout this paper are summarized in Table 1 for the ease of reference.

3.1 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 \( K = \{1, 2, \ldots, K\} \) 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 vMDC 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 vMDC 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 vMDC, for \( 0 < X_k < L_k \) and \( k \in K \).

With the partial computation offloading rule, every SMD requires a lot of physical resources, including computing, memory, storage, and network resources, to fulfill

² Due to the space limitation, specific detail about the vSMP Serve-rONE of the ScaleMP, Inc. is omitted here, and readers can refer to the website link (https://www.scalemp.com/products/vsmp-serve rone/) for more detailed description.
of required memory sizes in GB, respectively, to maintain the normal operation of PM m when it is idle, for m ∈ M.

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 the 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 \( \mathcal{N} = \{1, 2, \ldots, N\} \). We 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 \( \delta_k \ll N \). Denote by \( \Delta_k = \{1, 2, \ldots, \delta_k\} \) as the set of \( \delta_k \) VM instances.

---

In what follows, unless otherwise stated, we use the terms “computing resource” and “CPU kernels” interchangeably.
purchased by SMD \( k \), for \( \Delta_k \subset \mathcal{N} \). From the perspective of \( K \) SMDs, we can obviously have:
\[
\Delta_1 \cap \Delta_2 \cap \ldots \Delta_k \cdots \Delta_K = \emptyset,
\]
(1)
\[
\Delta_1 \cup \Delta_2 \cup \ldots \Delta_k \cdots \Delta_K = \mathcal{N}.
\]
(2)

Let us then characterize VM instance \( n \) as a virtual resource vector \( \Gamma_n = (c_n, s_n) \), 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:
\[
b_{n,m} = \begin{cases} 1, & \text{if VM instance } n \text{ is placed on PM } m, \ \forall n, \forall m, \\ 0, & \text{otherwise}. \end{cases}
\]
(3)

Thereby, elements \( c_n \) and \( s_n \) in virtual resource vector \( \Gamma_n \) of VM instance \( n \) can be respectively interpreted as:
\[
\sum_{m=1}^{M} b_{n,m} \alpha_{n,m} \leq c_n, \ \forall n \in \mathcal{N},
\]
(4)
\[
\sum_{m=1}^{M} b_{n,m} \beta_{n,m} \leq s_n, \ \forall n \in \mathcal{N}.
\]
(5)

### 3.2 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. 3. 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 (Zhang et al. 2018; Wang et al. 2018), 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. 3. Let \( t_k^0 \) 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 form \( K \) SMDs. Similar to (Zhang et al. 2018; Bi and Zhang 2018), 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.

Based on the time slot allocation as already introduced, we thereby 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. Here, the channel gain between SMD \( k \) and the gNodeB is denoted by \( g_k = g_k/d_k^{3/2} \), 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 (Yang et al. 2018), the small-scale fading channel gain is subject to the Rayleigh distribution, i.e., \( g_k \sim \mathcal{CN}(0, 1) \) with \( \mathcal{CN}(\nu_1, \nu_2) \) which is the complex Gaussian distribution with mean \( \nu_1 \) and variance \( \nu_2 \). 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 \( P_k \leq P_k^{\max} \). Also, the maximum transmit power \( P_k^{\max} \) (in Watt) of SMD \( k \) for uplink transmission can be calculated by:
\[
P_k^{\max} = 10^{0.1 P_0(dBm) - \theta \log d_k^\theta - 3},
\]
(6)
where \( P_0 \) is the receiving reference power by the gNodeB at a reference distance \( d_0 \). Based 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:

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

where \( \sigma^2 \) 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 \( p_k = \epsilon P_{k}^{\text{max}} \), where \( \epsilon \in (0, 1) \) refers to an adjustment factor.

### 3.3 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 \( \mu_k \) 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 \( \varphi_1 \) 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 \( \varphi_1 (L_k - X_k) \). Thus, the computation delay for processing the computation workload \( W_k \) locally can be calculated as:

\[
t^L_k = \frac{\varphi_1 (L_k - X_k)}{\mu_k f_k}.
\]

According to (Fan et al. 2007), we model the energy consumption of per CPU cycle at SMD \( k \) as \( \zeta_k (\mu_k f_k)^2 \), where \( \zeta_k \) 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:

\[
E^L_k = \varphi_1 \zeta_k (\mu_k f_k)^2 \left( L_k - X_k \right).
\]

#### 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. 3. With \( X_k \) bits workload offloaded to the vMDC, the total amount of uplink transmitted data for SMD \( k \) can be actually given by \( \tau X_k \), where \( \tau \) 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 is modeled as:

\[
t^O_k = \frac{\tau X_k}{R_k} = \frac{\tau X_k}{B \log_2 \left( 1 + \frac{p_k \eta_k}{\sigma^2} \right)}.
\]

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

\[
E^O_k = \frac{E^O_k}{t^O_k} = \frac{\epsilon \tau X_k P_{k}^{\text{max}}}{B \log_2 \left( 1 + \frac{p_k \eta_k}{\sigma^2} \right)}.
\]

#### 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 \( \rho \) 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 \( \mu_n \) CPU kernels under the virtualized MEC framework as already stated. Thus, the computing capability of VM instance \( n \) is specified by \( \rho \mu_n \) for \( n \in \mathcal{N} \). Then the computing capability of \( \delta_k \) VM instances purchased by SMD \( k \) is equal to \( \sum_{n \in \mathcal{N}_k} \rho \mu_n \). Let \( \varphi_2 \) 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:

\[
t^C_k = \frac{\varphi_2 \tau X_k}{\rho \mu_n} = \frac{\varphi_2 \tau X_k}{\sum_{n \in \mathcal{N}_k} \rho \mu_n} = \frac{\sum_{n \in \mathcal{N}_k} \rho \mu_n \alpha_{n,m}}{\sum_{n \in \mathcal{N}_k} \rho \mu_n \alpha_{n,m}}.
\]

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

\[
T^C = \sum_{k=1}^{K} t^C_k = \sum_{k=1}^{K} \sum_{n \in \mathcal{N}_k} \sum_{m=1}^{M} \frac{\varphi_2 \tau X_k}{\rho \mu_n \alpha_{n,m}}.
\]

We focus on the energy consumption in the computing stage in terms of the number of active PMs. According to
(Fan et al. 2007), as for the underlying PM, both the energy consumption and the CPU utilization rate 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:

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

Therefore, as in (Fan et al. 2007), by adopting its CPU utilization rate, the power consumption of PM $m$ can be defined as:

$$P_m = P_{\text{idle}}^m + (P_{\text{max}}^m - P_{\text{idle}}^m)y_m, \quad (15)$$

where $P_{\text{idle}}^m$ is the static power consumption when PM $m$ is idle and $P_{\text{max}}^m$ is the maximum power consumption when 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 (Fan et al. 2007). Hence, the energy consumption of $M$ PMs during the computing stage can be given by:

$$E^C = T^C \sum_{m=1}^{M} P_m = T^C \sum_{m=1}^{M} (P_{\text{idle}}^m + (P_{\text{max}}^m - P_{\text{idle}}^m)y_m)^{k} = \sum_{k=1}^{K} \sum_{m=1}^{M} \sum_{n=1}^{N} \frac{\varphi_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}} \left( \frac{M P_{\text{idle}}^m}{\sum_{n=1}^{N} b_{n,m} \alpha_{n,m}} \right), \quad (16)$$

where $E_{\text{idle}}^m = P_{\text{idle}}^m T^C$ is the energy consumption when PM $m$ is idle.

### 3.4 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:

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

where $w \in [0, 1]$ is the weighting factor reflecting the tradeoff between energy consumption and computing/offloading delay, $\chi_T$ is the normalization factor used to ensure the same range for two devised costs, $E_{\text{total}}$ is the total energy consumption cost, and $T_{\text{total}}$ 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:

$$E_{\text{total}} = \sum_{k=1}^{K} \left( E_{k}^1 + E_{k}^0 \right) + E^C$$

$$= \sum_{k=1}^{K} \left( \frac{\varphi_1 \tau X_k}{\mu k} \right) (L_k - X_k) + \frac{\alpha_k}{R_k} \sum_{n=1}^{N} a_{n,m} (p_{\text{max}}^m - p_{\text{idle}}^m) \frac{b_{n,m} \alpha_{n,m}}{a_m + u_m}. \quad (18)$$

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:

$$T_{\text{total}} = \sum_{k=1}^{K} (t_{e}^k + t_{o}^k + t_{c}^k)$$

$$= \sum_{k=1}^{K} \left( \frac{\varphi_1 (L_k - X_k)}{\mu k} + \frac{\tau X_k}{R_k} + \sum_{n \in \Delta_k} \sum_{m=1}^{M} \frac{\varphi_2 \tau X_k}{\rho b_{n,m} \alpha_{n,m}} \right). \quad (19)$$

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 underlying PMs. The optimization problem can be mathematically formulated as:

\begin{align*}
\min & \quad F(p_{n,m}) = w E_{\text{total}} + (1 - w) \chi_T T_{\text{total}} \\
\text{s.t.} & \quad \sum_{n=1}^{N} b_{n,m} \alpha_{n,m} \leq a_m + u_m \\
& \quad 0 \leq p_{n,m} \leq 1, \quad p_{n,m} \in \{0, 1\}.
\end{align*}
\[
\min_{b_{n,m}} \quad F(b_{n,m}) = wE_{\text{total}} + (1 - w)\chi_TT_{\text{total}}
\]  
(20)

\[\text{s.t.} \quad \sum_{m=1}^{M} b_{n,m}a_{n,m} \leq c_n, \quad \forall n, \]  
(21)

\[\sum_{m=1}^{M} b_{n,m}r_{n,m} \leq s_n, \quad \forall n, \]  
(22)

\[b_{n,m} \in \{0, 1\}, \quad \forall n, m. \]  
(23)

In (20), constraints (21) and (22) 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 (23) corresponds to the constraint of the binary placement variable, which also dictates whether VM instance is placed on top of the associated PM.

4 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 (20) as well as two linear constraints in (21) and (22). 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 using 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 (Manlove 2013). Inspired by (Bayat et al. 2016; Gu et al. 2015), in what follows, we will propose the effective algorithm based on two-sided matching theory (Manlove 2013; Bayat et al. 2016; Gu et al. 2015; Gao et al. 2017) 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).

4.1 Preference and preference list

The primary goal of the two-sided matching framework 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 (Bayat et al. 2016), 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 then 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 \(n \in \mathbb{N}\) and \(m \in \mathbb{M}\).

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:

\[\Psi_{n,m} = \frac{\alpha_m - \sum_{n=1}^{N} b_{(n-1),(m-1)}(n-1),(m-1) - \alpha_{n,m}}{\alpha_m + u_m}, \]  
(24)

\[Q_{n,m} = \frac{\beta_m - \sum_{n=1}^{N} b_{(n-1),(m-1)}(n-1),(m-1) - \beta_{n,m}}{\beta_m + v_m}, \]  
(25)

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 (24) and (25), 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 newly adopted PM is required to be turned on in an active mode for hosting the associated VM instances.

Accordingly, the preference matrix, denoted by \(Z_m\), between PM \(m\) and \(N\) VM instances can be specified by:

\[Z_m = \begin{bmatrix}
\Psi_{1,m} & \Psi_{2,m} & \ldots & \Psi_{n,m} & \ldots & \Psi_{N,m} \\
Q_{1,m} & Q_{2,m} & \ldots & Q_{n,m} & \ldots & Q_{N,m}
\end{bmatrix}^T, \]  
(26)
where \([\Psi_{1,m}, \Psi_{2,m}, \ldots, \Psi_{n,m}, \ldots, \Psi_{N,m}]\) are the preference values of the number of CPU kernels of VM instance \(n\) that is placed on PM \(m\), \([Q_{1,m}, Q_{2,m}, \ldots, Q_{n,m}, \ldots, Q_{N,m}]\) are the preference values of the amount of memory sizes of VM instance \(n\) that is placed on top of PM \(m\). Notation \([T]\) stands for the transpose of a matrix. 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 \(0 \leq q_1, q_2 \leq 1\). As a result, the preference vector \(\omega_m\) of PM \(m\) and \(N\) VM instances can be obtained as follows:

\[
\omega_m = \begin{bmatrix}
\Psi_{1,m} & \Psi_{2,m} & \ldots & \Psi_{n,m} & \ldots & \Psi_{N,m}
\end{bmatrix}^T, \quad \begin{bmatrix}
q_1 \\
q_2
\end{bmatrix}
\]

where \(\omega_{n,m}\) represents the preference degree of VM instance \(n\) that is placed on top of PM \(m\). From (27), 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
   (a) \(m\) is under subscribed and either of following three conditions is satisfied:
   (b) \(\mathcal{G}(n) \notin \mathcal{M}\), and \(n\) prefers \(m\) to \(\mathcal{G}(n)\);
   (c) \(\mathcal{G}(n) \notin \mathcal{M}\), and \(n\) is full and \(n\) prefers \(m\) to its current worst partner \(n_{\text{worst}}\);
4. \(m\) is full and \(m\) prefers \(n\) to its current worst partner \(n_{\text{worst}}\), 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 (Gale and Shapley 1962). 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 (Gale and Shapley 1962).

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. Having this in mind, from (24), preference list \(\mathcal{F}_m\) of PM \(m\) for \(N\) VM instances ranks each VM instance in this acceptable set \(\mathcal{W}_m = \{a_{1,m}, a_{2,m}, \ldots, a_{n,m}, \ldots, a_{N,m}|m \in \mathcal{M}\}\) 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 \(\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m, \ldots, \mathcal{F}_M\}\).

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 \(p\) 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 set $\mathcal{P} = \{\rho_1, \rho_2, \ldots, \rho_m, \ldots, \rho_M\}$. Meanwhile, the preference list of VM instance $n$ to $M$ PMs is defined as the computation capability of one CPU kernel of the PM, denoted by a set $S_n$. Preference list $S_n$ are arranged in descending order according to the CPU computing capability of PMs for each PM in this acceptable set $\mathcal{P} = \{\rho_1, \rho_2, \ldots, \rho_m, \ldots, \rho_M\}$, for $S_n \in \mathcal{P}$. The preference lists of $N$ VM instances is further denote by set $\mathcal{S} = \{S_1, S_2, \ldots, S_n, \ldots, S_N\}$.

### 4.2 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 (El-Atta and Moussa 2009) 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.

We then analyze the computational complexity of the proposed two-sided matching algorithm in Algorithm 1 based on the SPA. 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 $F_m$. 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 $S_n$ 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 $S_n$. 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 $O(M)$.

In summary, the overall computational complexity of Algorithm 1 for finding a two-sided matching between the VM instances and the PMs can be calculated as $O(M(N + M)) = O(M^2)$. Hence, the proposed two-sided matching algorithm only requires polynomial complexity to find the solution of the optimization problem in (20). We wish to further remark that the complexity of the proposed 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.

### 5 Simulation results and discussions

In this section, we present the simulation results to verify our theoretical analysis and evaluate the performance of our VM placement framework presented in this paper. Since it is extremely difficult to conduct repeatable and large-scale experiments for VM placement on a real 5G virtualized MEC platform, we thereby use a simulator developed in Python 3.7 as our simulation framework for performance evaluation. The simulations are run on a desktop PC with 4.80-GHz Intel Pentium Dual Core i7 CPU and 16 GB memory. All the simulation results are obtained in real time through our simulation framework.

As emphasized previously, the most prominent feature of the existing works associated with the VM placement for whether cloud data centers (Liu et al. 2018; Zhao et al. 2018; Dai et al. 2016; Rampersaud and Grosu 2017; Guo et al. 2018; Gharehpasha and Masdari 2020; Wei et al. 2020) or the virtualized MEC (Wang et al. 2017; Plachy et al. 2016; Zhao and Liu 2018) is that they only assume that a single VM instance must be placed on top of one PM via the Hypervisor or other virtualization software platform. However, our proposed VM placement framework 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 virtualized MEC system through the newly emerging high-performance virtualization platform, e.g., the vSMP ServerONE. Correspondingly, 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, and further gain insights into how the various system parameters affect the performance metrics in the simulations. Under the given simulation configurations, the performance of our proposed algorithm is numerically demonstrated and
discussed in terms of three mostly associated optimization metrics according to the idea of energy-delay tradeoff in this work, i.e., the total energy consumption, the total delay, and the total cost, respectively.

### 5.1 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 real-time simulations. We further assume that each SMD can purchase \(\delta_k = 4\) VM instances from the resourceful vMDC for computation offloading.

For simplicity, we only consider two types of physical resources for each PM, i.e., CPU kernels and memory sizes. Similar to many works that evaluated their algorithms or schemes by randomly generating VMs and PMs (Zhang et al. 2019a; Liu et al. 2018; Zhao et al. 2018), we also choose the same way throughout the simulations. More specifically, we randomly choose the number of VM instances hosted by the vMDC form the parameter range of \([200, 800]\). For convenience, we set the number of PMs randomly 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 specific resources for all the VM instances, i.e., CPU kernels and memory size.\(^4\) 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 Bayat et al. (2016); Rezvani et al. 2019) 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 \(\alpha_m\) and the amount of available memory sizes \(\beta_m\) 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 Bi and Zhang (2018); Chettri and Bera (2020); El-Atta and Moussa (2009), Filippou et al. (2020), 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.

In our simulations, the channel coefficients were generated following the quasi-static block fading channel model and the rest parameters were configured according to the settings of various works (Zhang et al. 2019a,2020c; Tan et al. 2018; Bi and Zhang 2018; Wang et al. 2019). All the simulation results obtained in this section are averaged over 1000 independently channel realizations. For clarity, the simulation configurations and detail parameter settings are

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

### Table 2 Simulation parameters

| Parameter | Description | Value |
|-----------|-------------|-------|
| \(P_0\)   | Receiving reference power by the gNodeB | 20 dBm |
| \(d_0\)   | Reference distance | 50 m |
| \(\theta\) | The path loss exponent | 2 |
| \(g_k\)   | Small-scale fading channel gain for SMD \(k\) | 30 |
| \(\sigma^2\) | Power of additive white Gaussian noise | \(-1777\) dBm |
| \(B\)     | Channel bandwidth | 400 MHz |
| \(d_k\)   | The distance from SMD \(k\) to the gNodeB | \([58, 59, 60, 65]\) m |
| \(\varepsilon\) | Adjustment factor | 0.9 |
| \(\mu_k\) | Number of CPU kernels for SMD \(k\) | Chettri and Bera (2020), El-Atta and Moussa (2009), Filippou et al. (2020) |
| \(f_k\)   | Local computation capability for SMD \(k\) | \(5.35 \times 10^8\) cycle/s |
| \(\tau\)  | Communication overhead | 100 bits |
| \(\varphi_1\) | CPU cycles of a bit at each SMD | 10 cycles/bit |
| \(\varphi_2\) | CPU cycles of a bit at the vMDC | 0.5 cycles/bit |
| \(\varsigma_k\) | Effective capacitance coefficient of SMD \(k\) | \(10^{-26}\) |
| \(L_k\)   | Total data size of workload for SMD \(k\) | \(1.37 \times 10^8\) bits |
| \(X_k\)   | Data size of offloaded workload from SMD \(k\) | \(1.25 \times 10^6\) bits |
| \(X_T\)   | Normalization factor | 1000 |
listed in Table 2, which did not change in the sequel unless explicitly stated otherwise.

5.2 Results

Figure 4 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 dedicated physical resources via our proposed two-sided matching algorithm. As shown in Fig. 4, the total energy consumption gradually tends to be the fixed constant value from $3.3878 \times 10^4$ to $1.6616 \times 10^4$ 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 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 dedicated physical resources via our proposed two-sided matching algorithm.

In Fig. 5, we compare the impact of the number of PMs on the total delay under the given number of VM instances. From Fig. 5, 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., $11.9989 - 11.0333$ 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., $5.5842 - 14.0755$ 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.

As depicted in Fig. 6, 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 form 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 $2.993 \times 10^4$ to $1.5582 \times 10^4$ 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.

As illustrated in Fig. 7, 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., from $8.8378 \times 10^4$ to $1.0273 \times 10^4$ 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. 8, we further study the effect of the varying number 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.

From the results of three sub-figures in Fig. 9, 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. 9 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 $7.9285 \times 10^{3}$ to $4.2922 \times 10^{4}$ 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.

Figure 10 plots the impact of both the number of PMs and the number of VM instances on the total cost under the different normalization factors with weighting factor $w = 0.5$.

Fig. 9  The impact of the number of VMs on the total cost

Fig. 10  The impact of both the number of PMs and the number of VMs on the total cost under the different normalization factors with weighting factor $w = 0.5$
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 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 $1.5514 \times 10^4$ to $7.928 \times 10^3$ with 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.

### 6 Summary

The performance of our VM placement framework introduced in Sect. 4 is numerically evaluated. Based on the simulation framework through our simulator developed in Python 3.7, we quantitatively demonstrate the effects of the number of PMs and the number of VM instances on the performance of the proposed matching algorithm. In particular, we focus on illustrating the total energy consumption, the total delay, and the total cost as the performance metrics. Furthermore, we obtain results illustrating the sensitivity of the system performance towards the choice of parameters in different scenarios, such as weighting factor $w$ and normalization factor $\chi_T$ in the weighted sum based cost function for computing and offloading.

The observations from the above analysis can be summarized as follows:

- With the increment of the number of PMs $M$, the total energy consumption takes a substantially downward trend in the VM placement framework under the given number of VM instances $N$ when $M \leq 150$.
- The total delay shows a slowly decrease trend with the growing number of PMs from $M = 50$ to $M = 150$. That is, to say, 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.
- For the same number of VM instances $N$, the total cost obviously decreases with the number of PMs $M$ increasing for the virtualized MEC system. Moreover, the total cost under the smaller weighting factor $w$ is obviously smaller than that of the larger one.
- For the same number of PMs $M$, the total energy consumption with the smaller number of VM instances $N$ outperforms the total energy consumption using the larger number of VM instances $N$.
- With the increment of the number of VM instances $N$, the total delay takes an upward trend for the same number of PMs $M$.
- The total cost lies in a continuous growth when the number of VM instances increases from $N = 200$ to $N = 800$. More particularly, when the weighting factor $w$ continuously magnifies, the total cost takes an upward trend for the VM placement framework.
- For the same weighting factor $w$, the larger the normalization factor $\chi_T$, the higher the total cost for our proposed VM placement framework in the virtualized MEC system under the constraint of the given number of PMs $M$ and the given number of VM instances $N$.

### 7 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.

### Acknowledgements

This work was supported in part by the National Natural Science Foundation of China (61971032, 61901115), the Natural Science Foundation of Hebei Province (F2019402206, A2020402013, F2018402198), the Research Program for Top-notch Young Talents of Hebei Province (BJ2017037), the Scientific Research Foundation of Higher Education of Hebei Province (QN2019059), the Natural Science Foundation of Shaanxi Province (2020JM-361), and the Fundamental Research Funds for Central Universities (FRF-TP-18-008A3, 3282019111).
References

Bayat S, Li Y, Song L, Han Z (2016) Matching theory: applications in wireless communications. IEEE Signal Process Mag 33(6):103–122

Bi S, Zhang YJ (2018) Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans Wireless Commun 17(6):4177–4190

Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2019) Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J 6(3):4005–4018

Chetti L, Bera R (2020) A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. IEEE Internet Things J 7(1):16–32

dai X, Wang JM, Bensaou B (2016) Energy-efficient virtual machines scheduling in multi-tenant data centers. IEEE Trans on Cloud Comput 4(2):210–221

El-Atta AHA, Moussa MI (2009) Student project allocation with preference lists over (student, project) pairs. In: 2009 International Conference on Computer and Electrical Engineering, Dubai, United Arab Emirates, 28 December-30 December. IEEE, pp. 1–6

Fan X, Weber WD, Barroso LA (2007) Power provisioning for a warehouse-sized computer. In: 2007 ACM International Symposium on Computer Architecture (ISCA), San Diego, CA, USA, 9 June-13 June. ACM, pp. 13–23

Filippou MC, Sabella D, Emara M, Prabhabakan S, Shi Y, Bian B, Rao A (2020) Multi-access edge computing: A comparative analysis of 5G system deployments and service consumption locality variants. IEEE Commun Standards Mag 4(2):32–39

Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69(1):9–15

Gao L, Duan L, Huang J (2017) Two-sided matching based cooperative spectrum sharing. IEEE Trans Mobile Comput 16(2):538–551

Gharehpasha S, Masdari M (2020) A discrete chaotic multi-objective SCA-ALO optimization algorithm for an optimal virtual machine placement in cloud data center. J Amb Intel Hum Comp 12:1–17

Gu Y, Saad W, Bennis M, Debkah M, Han Z (2015) Matching theory for future wireless networks: fundamentals and applications. IEEE Commun Mag 53(5):52–59

Guo Y, Stolyar AL, Walid A (2018) Shadow-routing based dynamic algorithms for virtual machine placement in a network cloud. IEEE Trans Cloud Comput 6(1):209–220

Hawilo H, Jammal M, Shami A (2019) Network function virtualization-aware orchestrator for service function chaining placement in the cloud. IEEE J Sel Areas Commun 37(3):643–655

Kulkarni SG, Liu G, Ramakrishnan KK, Arumaiturai M, Wood T, Fu X (2020) REINFORCE: achieving efficient failure resiliency for network function virtualization-based services. IEEE/ACM Trans Netw 28(2):695–708

Li D, Hong P, Xue K, Pei J (2019) Virtual network function placement and resource optimization in NFV and edge computing enabled networks. Comput Netw 152:12–24

Li L, Quck TQS, Ren J, Yang HH, Chen Z, Zhang Y (2021) An incentive-aware job offloading control framework for multi-access edge computing. IEEE Trans Mobile Comput 20(1):63–75

Liang Z, Liu Y, Lok TM, Huang K (2019) Multiuser computation offloading and downloading for edge computing with virtualization. IEEE Trans Wireless Commun 18(9):4298–4311

Liu XF, Zhan ZH, Deng JD, Li Y, Gu T, Zhang J (2018) An energy efficient ant colony system for virtual machine placement in cloud computing. IEEE Trans Evol Comput 22(1):113–128

Manlove DF (2013) Algorithmics of matching under preferences. World Scientific

Mao Y, Zhang J, Letairy KB (2016) Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J Sel Areas Commun 34(12):3590–3605

Plachy J, Becvar Z, Strinati EC (2016) Dynamic resource allocation exploiting mobility prediction in mobile edge computing. In: 2016 IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Valencia, Spain, 4 September-8 September. IEEE, pp 1–6

Rampersaud S, Grosu D (2017) Sharing-aware online virtual machine packing in heterogeneous resource clouds. IEEE Trans Parallel Distrib Syst 28(7):2046–2059

Rezvani S, Farsaeeafard S, Mokari N, Javan MR, Yanikomeroglu H (2019) Cooperative multi-birate video caching and transcoding in multicarrier NOMA-assisted heterogeneous virtualized MEC networks. IEEE Access 7:93511–93536

Strinati EC, Barbarossa S, Gonzalez-Jimenez J, Ktenas D, Cassina M, Maret L, Dehos C (2019) 6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication. IEEE Veh Technol Mag 14(3):42–50

Sun Z, Nakhai MR (2020) Distributed mirror-prox optimization for multi-access edge computing. IEEE Trans Commun 68(5):3096–3106

Tan Z, Yu R, Li X, Ji H, Leung VCM (2018) Virtual resource allocation for heterogeneous services in full duplex-enabled SCNs with mobile edge computing and caching. IEEE Trans Veh Technol 67(2):1794–1808

Wang W, Zhao Y, Tornatore M, Gupta A, Zhang J, Mukerjee B (2017) Virtual machine placement and workload assignment for mobile edge computing. In: 2017 IEEE International Conference on Cloud Networking (CloudNet), Prague, Czech Republic, 25 September-27 September. IEEE, pp 1–6

Wang F, Xu J, Wang X, Cui S (2018) Joint offloading and computing optimization in wireless powered mobile-edge computing systems. IEEE Trans Wireless Comput Netw 17(3):1784–1797

Wang Y, Tao X, Zhang X, Zhang P, Hou YT (2019) Cooperative task offloading in three-tier mobile computing networks: An ADMM framework. IEEE Trans Veh Technol 68(3):2763–2776

Wei W, Wang K, Wang K, Gu H, Shen H (2020) Multi-resource balance optimization for virtual machine placement in cloud data centers. Comput Electr Eng 88:106866

Yang Z, Ding Z, Fan P, Karagiannidis GK (2018) On the performance of non-orthogonal multiple access systems with partial channel information. IEEE Trans Commun 66(2):654–667

Yang S, Li F, Trajanovski S, Yahyapour R, Fu X (2021) Recent advances of resource allocation in network function virtualization. IEEE Trans Parallel Distrib Syst 32(2):295–314

Yi B, Wang X, Li K, Das SK, Huang M (2018) A comprehensive survey of network function virtualization. Comput Netw 133:212–262

Zhang L, Zhao Z, Wu Q, Hao H, Xu H, Wu X (2018) Energy-aware dynamic resource allocation in UAV assisted mobile edge computing over social internet of vehicles. IEEE Access 6:56700–56715

Zhang L, Zhang H, Yu L, Xu H, Song L, Han Z (2019a) Virtual resource allocation for mobile edge computing: A hypergraph matching approach. In: 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9 December-13 December. IEEE, pp 1–6

Zhang L, Zhao H, Hou S, Zhao Z, Xu H, Wu X, Wu Q, Zhang R (2019b) A survey on 5G millimeter wave communications for UAV-assisted wireless networks. IEEE Access 7:117460–117504

Zhang H, Zhang J, Long K (2020a) Energy efficiency optimization for NOMA UAV network with imperfect CSI. IEEE J Sel Areas Commun 38(12):2798–2809
Zhang L, Hu J, Guo C, Xu H (2020b) Dynamic power optimization for secondary wearable biosensors in e-healthcare leveraging cognitive WBSNs with imperfect spectrum sensing. Future Gener Comput Syst 112:67–92
Zhang L, Zhang G, Zhao X, Li Y, Huang C, Sun E, Huang W (2020c) Resource allocation for energy efficient user association in user-centric ultra-dense networks integrating NOMA and beamforming. AEU Int J Electron Commun 124:1–19
Zhao L, Liu J (2018) Optimal placement of virtual machines for supporting multiple applications in mobile edge networks. IEEE Trans Veh Technol 67(7):6533–6545
Zhao H, Wang J, Liu F, Wang Q, Zhang W, Zheng Q (2018) Power-aware and performance-guaranteed virtual machine placement in the cloud. IEEE Trans Parallel Distrib Syst 29(6):1385–1400

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.