Infrastructure-efficient Virtual-Machine Placement and Workload Assignment in Cooperative Edge-Cloud Computing over Backhaul Networks

Wei Wang[1], Massimo Tornatore[2, 3], Yongli Zhao[1], Haoran Chen[1], Yajie Li[1], Abhishek Gupta[2], Jie Zhang[1], and Biswanath Mukherjee[2]

Abstract—Edge computing provides computing capability at close-user proximity to reduce service latency for end users. To improve the efficiency of edge computing infrastructures, geographically-distributed edge datacenters can co-work with each other and with cloud datacenters, forming a new paradigm referred to as cooperative edge-cloud computing. In this context, applications typically run on a virtual machine (VM) that can be replicated at multiple sites, and thus user traffic can be served at all the sites where corresponding VMs reside.

For the performance of many applications, latency is a critical parameter. In this work, taking applications' latencies as the primary constraint, we model the problem of "VM placement and workload assignment" as a mixed integer linear program and develop heuristic algorithms accordingly. The goal is to minimize the consumption of information technology (IT) infrastructures for placing VMs in cooperative edge-cloud computing, while meeting the heterogeneous latency demands of different applications. Some preliminary results indicate that edge datacenter's resource efficiency can be optimized by proper cross-site VM placement and workload re-direction.

***Index Terms*—edge and cloud computing, latency, virtual machine, workload, backhaul networks.**

I. INTRODUCTION

CLOUD computing is a popular paradigm for application provisioning, and the datacenter (DC) represents the primary information technology (IT) infrastructures to provide hardware resources (computing, network, and storage). DCs in cloud computing are usually centralized, i.e., located at few sites, which could be far away from most end-user equipment. As a result, user equipment may experience longer latency, due to both the longer distance to be traveled to reach cloud DC and extra transmission latency in case of high network load. Recently, new emerging applications (e.g., self-driving cars) with ultra-low latency demands have emerged, and cloud computing may not be able to directly serve these applications due to excessive latency. In this case, edge computing has been introduced to serve users at close proximity [1, 2].

This work is an extended version of a published conference paper [33].

Wei Wang, Yongli Zhao*, Haoran Chen, Yajie Li, and Jie Zhang are with Beijing University of Posts and Telecommunications, China. (e-mail: {weiw, yonglizhao, haoranc, yajieli, lgr24}@bupt.edu.cn).

Massimo Tornatore, Abhishek Gupta, and Biswanath Mukherjee are with University of California, Davis, USA (e-mail: abgupta@ucdavis.edu, bmukherjee@ucdavis.edu). Massimo Tornatore is also with Politecnico di Milano, Italy (e-mail: massimo.tornatore@polimi.it).

Edge computing infrastructures are usually deployed within the footprint of access-aggregation networks, in the form of small DCs, called edge DCs [3]. The edge DCs can take parts of the jobs of user-equipment (*task offloading*) [4] to relieve their battery and computation-power limitations. Edge DCs can also take parts of the cloud's jobs to serve requests from user equipment (*edge-based service*) and thus avoid long-distance transmission to the cloud. In the latter case, due to the limited hardware capacity, an edge DC might not be able to serve a massive number of application requests from a local area, especially at internet peak/rush hours. Hence, also considering that different applications have heterogeneous latency demands, some traffic can still be efficiently served in a farther edge/cloud DC as long as their latency demands are satisfied [5]. That is, besides serving local traffic, edge/cloud DCs can also serve traffic from other sites [6]. Such a paradigm is referred to as *cooperative edge-cloud computing*. In this paradigm, edge and cloud DCs are inter-connected by backhaul networks, which comprise aggregation networks and cross-region backbone networks. From the infrastructure point of view, the backhaul networks between edge and cloud DCs are mostly powered by optical networks for aggregation and transport [7]. As the essential part to provide network connectivity for edge-cloud cooperation, the backhaul networks have significant influence on the quality of experience (QoE) in the context of edge computing.

In DCs, virtual machine (VM) is now a mature technique for service provisioning. A VM uses only a subset of a server's hardware and can still support the processing and data storage required by an application. Proper placement of VMs in cooperative edge-cloud computing is crucial for satisfactory QoE. To manage VMs in edge-cloud DCs, IT operators and service providers must address a "VM placement and workload assignment" problem, i.e., they must decide in which sites to place VMs and the number of VMs for each application at each edge/cloud DC (VM placement), and reserve VM processing capacity for given application requests (workload assignment).

Both VM placement and workload assignment have strong impacts on service latency: *i*) VM placement sets the network distance between users and VMs, hence impacting latency; *ii*) workload assignment affects the processing time of a request since we can reduce queuing and processing time by assigning more processing capacity from VMs to an application request to avoid congestion. Intuitively, placing more VMs can decrease service latency by alleviating congestion. However, each edge DC has finite capacity, limiting the number of VMs.

Targeting the general edge-based service paradigm instead of a specific use-case in edge-cloud computing, we further formulate a mixed integer linear program (MILP) and propose heuristic algorithms for placing VMs and assigning workloads in a backhaul-networked cooperative edge-cloud computing system. The goal is to improve the IT infrastructure efficiency by minimize hardware consumption, while meeting latency demands of different applications. The preliminary numerical results indicate some of the relevant factors that will affect the hardware consumption of DCs.

II. Related Works

In the context of edge-cloud computing, many works studied task offloading issues. In [8], considering the fact that wireless channels shared among mobile devices and edge servers also constrain task offloading decisions, a programming problem was formulated to minimize the average application response time by scheduling offloading tasks and allocating bandwidth jointly. From another perspective, in [9] the authors observed that mobile devices' CPU frequency has significant impact on task offloading and proposed an approach to minimize tasks' execution latency and energy consumption by optimizing the task and CPU frequency allocation decisions. In [10], taking both mobile devices' CPU frequency and wireless channels into account, an algorithm that can select the CPU frequency, the tasks to be offloaded, and the network path, was designed to reduce the total energy cost and execution delay. As discussed, task offloading and edge-based service are two different paradigms in edge computing, and corresponding research topics are different too. To save space, the following sub-sections will mainly focus on the existing works about edge-based service, which is more relevant to this work.

A. VM Placement and Workload Management in Cloud

VM placement is a well-studied research topic in IT resource management field. VM placement strategies (e.g., VM mapping and VM migration) have been intensively studied to find an optimal physical machine to host a VM and/or to migrate an existing VM [11]. A popular topic in VM placement is minimization of energy and resource utilization [12-14]. Ref. [12] reviewed most existing works on energy-efficient VM management strategies in cloud computing, and survey [15] covered recent topics on load balancing in cloud computing. In particular, for the multi-site cloud scenario, global server load balancing (GSLB) is a popular solution for orchestrating traffic over multiple datacenters [16]. The multi-site cloud architecture of GSLB is similar with the cooperative edge-cloud in this work, but the edge-cloud scenario has brought new challenges (i.e., edge DCs' capacity limitation and its non-linear impact on service latency), making the traffic management issue in edge-cloud different with that for GLSB.

B. VM placement and workload management in Edge

VM placement and workload management are well studied in cloud computing. However, edge computing introduces new characteristics [17], and the challenges for VM placement and workload management in edge computing are not the same as that in cloud computing. First, the capacity of cloud DCs is rarely discussed, as it is usually assumed as sufficient. In edge computing, the number of distributed edge DCs can be large, but their capacity is limited, to reduce cost [18]. So, new challenges arise related to the limited capacity of edge DC to provide better QoE. Also, applications in cloud-computing were never as latency-sensitive as those in edge computing. Even though, the network transporting traffic to the cloud might be optimized in terms of latency [19], it has been shown that latency requirements cannot be satisfied using traditional cloud architectures. When dealing with ultra-low latency applications (e.g., those in the area of Tactile Internet), edge-cloud service providers are hence used to shorten the service latency by reducing the geographical distance and deploying edge datacenters [20, 21]. That is, edge DC's location and distance have significant impact on latency, and they must be dealt with carefully. In addition, compared with cloud computing, edge computing is context-aware, and thus edge computing needs to take care of the context-related factors such as mobility. Focusing on user mobility in mobile networks, Ref. [22] studied the problem of VM scheduling to migrate VMs as user's locations change. Beyond VM scheduling, Ref. [23] proposed a mobility-aware service placement framework to migrate VMs in real-time without priori-knowledge of user mobility, while balancing the migration cost and QoE degradation. Targeting the tradeoff between latency of edge and high energy efficiency of cloud, Ref. [24] studied workload management in edge computing, and formulated a model to balance latency and power consumption. Besides placing VMs on given edge/cloud DCs, Ref. [25] also studied the cloudlet placement issue for the fiber-wireless-backed edge computing, and presented an integer linear program model to optimize the deployment cost, while meeting stringent response time of mobile users.

C. Our contributions

To cope with the incoming deployment of ultra-low latency services, infrastructure providers who are operating edge DCs must adopt effective resource-allocation strategies to better utilize edge DC's limited capacity while meeting the required service latency. In edge context, the existing VM/applications placement works have been summarized in survey [26]. In the following, we selectively overview the relevant contributions in the area of resource allocation and workload management.

Based on the limitations of edge computing resources, authors in [27] investigated the problem of jointly performing load distribution and placement of edge-based services. For this problem, an integer nonlinear programming model and two algorithms to minimize the violation of QoE constraints were proposed. Ref. [28] further incorporated the possibility of cooperation among multiple edge DCs, and proposed a decentralized algorithm to collaboratively place services over edge DCs with the aim to minimize the traffic load caused by forwarding traffic across multiple edge DCs. Ref. [29] studied a similar service placement issue in edge, but its objective is to maximize the utilization of edge nodes. In [30], authors considered that resource usage changes over time (e.g., daily fluctuation), and proposed an integer nonlinear programming

model and two algorithms to deploy and release edge services dynamically while minimizing the cost for service provisioning and meeting QoE constraints. Further incorporating VMs as the middleware between hardware and applications, authors in [31] studied the problem of VM placement for multiple applications, and proposed heuristic VM placement algorithms to minimize the average response time. In [32], deep reinforcement learning was employed for efficient and adaptive allocation of computing and network resources, with the goal of reducing average service time and evenly assign resource usage.

Also considering multiple application's heterogeneous latency requirements, instead of taking latency minimization as the primary objective, our work in [33] studied the hardware consumption minimization issue with application's latency requirements and infrastructure capacity as constraints, and formulated the VM placement and workload assignment problem as a MILP model to minimize the consumed hardware resources in edge/cloud DCs for placing application VMs. From another perspective, the infrastructure efficiency is improved if we can reduce consumed hardware resources (in short, hardware consumption) for placing VMs that are used to serve given workloads. Since the MILP model usually has scalability issues, in this work we extend the work in [33], and we also provide heuristic approaches for placing VMs and for assigning workload to support multiple applications.

Among above related works, this work is more similar with two works. First, the hardware consumption minimization issue in this work sounds like the resource cost minimization issue in [30], but they are actually different. The resource cost in [30] was studied in the context of dynamic service provisioning, and thus it refers to the cumulative cost of dynamically allocated network/ computing resources for serving given traffic; while our hardware consumption refers to the hardware resources for placing VMs that are used to serve estimated traffic, in a resource planning context. More specifically, with VM as the middleware between hardware and application service, the consumed hardware in our work is not proactively allocated/ released. Second, the VM-based and multi-applications context in this work is very similar with that in [31], but their objectives are completely different, i.e., this work focuses on hardware consumption minimization, while Ref. [31] focuses on average latency minimization.

In summary, the new contributions in this work, compared with related works and previous conference version, includes: 1) the heuristics for placing VMs and assigning workloads for multiple applications; 2) the comparison of the models with/without small flows (see III.C) and 3) the preliminary running-time/complexity comparison for the approaches.

III. VM Placement and Workload Assignment

This section discusses the latency components of edge-based services and introduces the problem of VM placement and workload assignment. We will use term DC to represent both an edge DC and a cloud DC, and a remote DC refers to either a remote edge DC or a cloud DC. Also, we will abbreviate term "application" as "app" in the following.

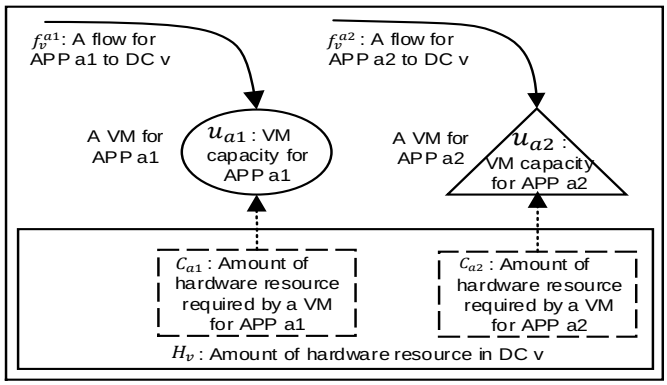

Fig. 1. Relationship among flows, VMs, and hardware resources in DCs.

A. Overview of Cooperative Edge-Cloud System

Basically, edge DCs could be deployed at any network node within access/aggregation networks. One popular use-case is that edge DCs are co-located with aggregation points (APs) (e.g., wireless baseband processing unit) in access networks, and each edge DC covers a group of end users. Although edge DCs are primarily used for serving local users, they are also reachable by remote users through the network. We assume that requests from each user equipment (UE) must go through corresponding local AP for aggregation, the UE-AP connections can be removed from our modeling and the edge DC that is co-located with AP can be modelled as the source of each request. In such a model, we call the requests for app a, originating from edge DC v, as a flow f_v^a (note that a flow is a set of requests that are aggregated from many local UEs attached to the same AP) We statistically model the arrival of individual requests (our model considers Poisson arrivals), and use r_v^a to represent the arrival-rate (workload) of flow f_v^a in terms of the number of arriving requests per second.

Let A denote a set of apps. Each app $a \in A$ is supported by a unique type of VM (i.e., application-specific VM). Fig. 1 shows the serving relationship among flows, VMs, and hardware resources in a DC. For DC v, let H_v denote its overall hardware resource capacity. Inside a DC, each VM requires a certain amount of hardware resources (composed of CPU, memory, and storage) for self-running and service provisioning. For each app a, we define C_a as the amount of hardware resources required to run a corresponding VM, and u_a as the amount of processing capacity a VM has for app a. The u_a is measured by the number of requests a VM can process per second, and we also call u_a as *VM capacity* of a VM for a. Without losing generality, C_a and u_a are assumed to be fixed. In addition, each app a is characterized by a maximum tolerated latency T_a.

B. Latency Analysis

Service latency refers to the time from sending an app request to receiving a response; and it is composed of transmission, propagation, queuing, and processing times.

In edge-cloud computing, user's requests might be served by a local or remote DC. Fig. 2 shows the latency components arising in (a) local and (b) remote processing scenarios. From UE to AP/edge-DC, each request experiences a certain transmission latency at UE and a certain propagation latency on

UE-AP network connection. Fig. 2(a) shows the local processing scenario, where requests are served directly by local edge DC, i.e., queuing and processing latencies occur at local edge DC. If local edge DC is unable to serve a request, the request will be re-directed to a remote edge/cloud DC via backhaul networks, as shown in Fig. 2 (b). In this case, the data transmission over backhaul networks between source and destination DCs will cause additional network latency, while queuing and processing latencies occur at the remote DC.

Since UE and UE-AP connection are not affected by VM placement and workload assignment, UE-AP side latency cannot be optimized in this work. Thus, this work focuses on the two latency components that can be optimized by proper VM placement and workload assignment: inter-DC backhaul network latency, and queuing and processing latency at DCs[1].

Propagation latency between DCs, can be assumed with very diverse values, depending on the specific network connectivity between related DCs. Let V denote a set of DCs. Since the backhaul infrastructure among edge and cloud DCs are quite diverse for a general modeling, we simplify the network propagation latency $t_{s,d}$ between DC pair (s,d) by assuming it is proportional to the distance between s and d.

Queuing and processing latencies in the destination DC, are determined jointly by the offered workload and allocated VM capacity. In a practical system, the VM side latency model is quite complex, and it could be impacted by many factors [34]. For this study, we choose the classical M/M/1 queue model as an example to estimate the processing and queuing latency at a VM (this model could be replaced with more complex/accurate models such as M/M/S, and the proposed framework will still be applicable). Hence, given a flow, whose workload is r, and assuming the VM capacity assigned to r is u, the average queuing and processing latency is as Eqn. (1).

Of course, many other factors in backhaul networks (such as optical-electrical conversion at intermediate nodes) might also have influence on the overall service latency. But the latency at intermediate nodes can be omitted safely if we consider the optical backhaul, where most parts of the inter-DC connectivity are buffer-less optical channels. With regard for the fact that latency is the primary constraint for the QoE in edge computing, the specific network topology, which may affect the cross-DC routing for dynamic service provisioning, is simplified as a set of inter-DC latencies, with the assumption that backhaul network's bandwidth capacity is sufficient for the inter-DC cooperation in edge-cloud computing.

C. VM Placement and Workload Assignment

To deploy services for apps, edge-cloud DC operators have to place VMs and assign VM capacities to given traffic flows, while meeting apps' heterogeneous latency requirements.

As for VM placement, we must consider that each DC has finite hardware capacity, and this limits the number of VMs a DC can host. H_v, the amount of hardware resources in DC v, together with C_a, the hardware required by a single VM for app a, set the number of VMs for each app.

As for workload assignment, we must consider that according to Eqn. (1), the more VM capacity allocated to a flow, the less time is needed for processing and queuing. Hence, the VM capacity to be allocated to a flow depends on how much time is available for queuing and processing.

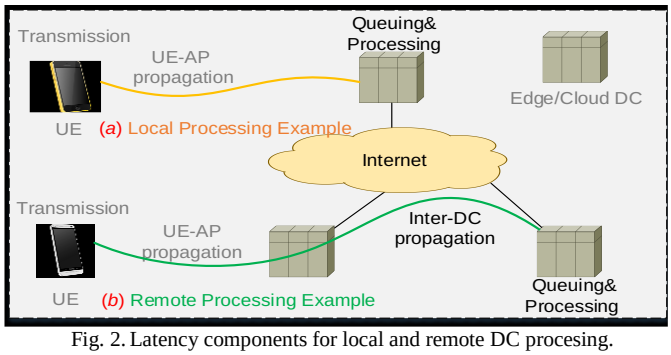

Fig. 2. Latency components for local and remote DC procesing.

To provide finer granularity for managing request flows (especially useful in case of congestion), we also consider that one flow can be split into two types of sub-flows (i.e., dividing requests in a flow into different groups for separate processing), and each sub-flow can be served separately in different VMs. Each sub-flow can be a *big flow* or a *small flow* (defined later). For a given flow, its majority part, which requires the full capacity of multiple individual VMs, is referred to as a *big flow*. Let $b_{s,d}^a$ denote the workload of the big flow f_s^a that is assigned to DC d. Assuming $b_{s,d}^a$ is assigned to $m_{s,d}^a$ VMs at DC d and is dispatched to VMs evenly, the average queuing and processing latency for $b_{s,d}^a$ is calculated as Eqn. (2). For the same given flow, the left part (if any), which requires only parts of the capacity of one individual VM, is referred to as a *small flow* (this is an optional definition for extra optimization, not essential to our model). Let $s_{s,d}^a$ denote the workload of a small flow f_s^a that is assigned to DC d. Assuming the VM capacity assigned to $s_{s,d}^a$ is $u_{s,d}^a$, the average queuing and processing latency for $s_{s,d}^a$ can be calculated as in Eqn. (3). Note that one can use other methods to dispatch workloads. Note also that a VM is sharable only to small flows, but not to big flows.

$$t = 1/(u-r) \tag{1}$$

$$p_{s,d}^a = 1/(u_a - b_{s,d}^a / m_{s,d}^a) \tag{2}$$

$$p_{s,d}^a = 1/(u_{s,d}^a - s_{s,d}^a) \tag{3}$$

$$t_{s,d}^a = p_{s,d}^a + 2t_{s,d} \tag{4}$$

$$m_{s,d}^a = b_{s,d}^a / (u_d - 1/(t_{s,d}^a - 2t_{s,d})) \tag{5}$$

In summary, the queuing and processing latency for each sub-flow (small or big flow) can be calculated as in Eqn. (2) or (3). The overall service latency for each sub-flow must also consider the inter-DC latency $t_{s,d}$, representing the network latency between source DC s and destination DC d, and thus the overall service latency would be Eqn. (4). Note that $t_{s,d} = 0$ when $s = d$, which means a flow is assigned to VM(s) at the source DC for local processing. According to Eqns. (3) and (4), the minimum number of required VMs for a big flow $b_{s,d}^a$ at DC d is calculated as Eqn. (5).

[1] Compared with the app layer service time, the queuing and processing time at intermediate routers/switches are assumed to be much less, so this work will mainly focus on queuing and processing latency at destination VMs.

IV. Mixed Integer Linear Program (MILP) Model for Hardware Consumption Minimization

This section formulates the problem of VM placement and workload assignment, with the objective to minimize hardware consumption. Hardware consumption means the cost of leasing space to place VMs for service providers, and it also means resource efficiency for infrastructure operators. The model turns out to be a mixed integer linear program (MILP).

A. Problem Description

Given a set of DCs, hardware capacity of each DC, network latency between each DC pair, a set of apps, hardware resources required to deploy a VM for each app, capacity of a VM and maximum tolerated latency for each app, and estimated workloads of each flow, we determine the number of VMs for each app at each DC, the workload, destination, and VM capacity for each sub-flow, subject to the constraints below. Note that all the variables in this model are non-negative.

B. Input Parameters

V: Set of DCs

A: Set of apps

H_v: Hardware resource capacity in DC v, where $v \in V$

$t_{s,d}$: Propagation latency from DC s to DC d, where $s, d \in V$

T_a: Tolerated latency threshold for app a, where $a \in A$

C_a: Amount of hardware resources required by a VM for app a

u_a: VM capacity of one VM for app a

r_v^a: Workload of a flow that originates from DC v for app a

C. Variables

n_v^a: Integer, number of VMs for app a placed at DC v

$b_{s,d}^a$: Float, workload of big flow, for app a, from DC s to d

$m_{s,d}^a$: Integer, number of VMs allocated for $b_{s,d}^a$

$s_{s,d}^a$: Float, workload of small flow, for app a, from DC s to d

$g_{s,d}^a$: Binary, a flag indicating whether $s_{s,d}^a$ is zero or not

$u_{s,d}^a$: Float, allocated VM capacity to $s_{s,d}^a$

D. Objective: Minimize Hardware Consumption

$$\text{Min} \sum_{v \in V} \sum_{a \in A} n_v^a C_a \quad (6)$$

E. Constraints

$$\sum_{a \in A} n_v^a C_a \le H_v \qquad v \in V \quad (7)$$

$$\sum_{d \in V} b_{s,d}^a + s_{s,d}^a = r_s^a \qquad s \in V,\ a \in A \quad (8)$$

$$\sum_{s \in V} u_{s,d}^a + u_a m_{s,d}^a = u_a n_d^a \qquad d \in V,\ a \in A \quad (9)$$

$$b_{s,d}^a \le (u_a - \frac{1}{(T_a - 2t_{s,d})}) m_{s,d}^a \quad (s,d) \in V, a \in A \quad (10)$$

$$s_{s,d}^a \le u_{s,d}^a - g_{s,d}^a / (T_a - 2t_{s,d}) \quad (s,d) \in V, a \in A \quad (11)$$

$$u_a m_{s,d}^a - b_{s,d}^a \ge 0 \quad (s,d) \in V, a \in A \quad (12)$$

$$u_{s,d}^a - s_{s,d}^a \ge 0 \quad (s,d) \in V, a \in A \quad (13)$$

$$(T_a - 2t_{s,d}) m_{s,d}^a \ge 0 \quad (s,d) \in V, a \in A \quad (14)$$

$$(T_a - 2t_{s,d}) g_{s,d}^a \ge 0 \quad (s,d) \in V, a \in A \quad (15)$$

$$g_{s,d}^a = 0 \Leftrightarrow s_{s,d}^a = 0 \quad (s,d) \in V, a \in A \quad (16)$$

Objective (6) calculates total required hardware resources of all VMs for all apps in all DCs, and this model tries to minimize it. Constraint (7) guarantees that the hardware resources occupied by the VMs at each DC cannot exceed the host DC's hardware capacity. Eqn. (8) enforces that each flow is totally assigned, in forms of either small sub-flows or big sub-flows. Eqn. (9) enforces that for all small flows and big flows, the allocated VM capacity cannot exceed the overall capacity of VMs at target DC. Constraints (10) and (11) are transformed from Eqn. (4), guaranteeing that service latency of both big flows and small flows are within the corresponding app's latency requirement. Constraints (12) and (13) guarantee that the queuing systems of all flows are stable by assuring that the assigned workload for each flow cannot exceed the allocated VM capacity. Constraints (14) and (15) are two auxiliary constraints for (10) and (11), as (10) and (11) do not apply when $(T_a - 2t_{s,d})$ (i.e., time left for queuing and processing) is negative. They guarantee that both big and small flows are not routed to a DC whose latency to the flow's source DC exceeds their latency thresholds. Note that since we are studying the workload assignment problem for flows (comprise of multiple requests) instead of individual requests, the latency threshold here refers to the maximum of the average latency of requests in a flow. Constraint (16) enforces the relation between $s_{s,d}^a$ and its binary flag $g_{s,d}^a$. Note that this model can result as infeasible if no DC is able to provide sufficient hardware resources to meet the latency threshold of a pending flow.

V. Heuristic Algorithm for VM Placement and Workload Assignment

As MILP has scalability issues for large-scale problems, we also develop a heuristic algorithm to get acceptable and efficient solutions for VM placement and workload assignment.

The goal of the proposed heuristic is to decide the number of VMs for each app at each DC, and assign each flow to placed VMs. In general, VM placement and workload assignment are two interdependent problems. On one hand, VM placement is driven by given workloads; on the other hand, workload assignment depends on the capacities of the placed VMs. Therefore, we cannot simply place VMs first, and then assign workload to the VMs. Instead, this work tries to solve this problem by placing VMs and assigning workloads iteratively.

Since overall DC hardware resources are shared by multiple workload flows, the sorting order of these flows has a strong impact on total hardware consumption. To reduce the impact of sorting order, we consider further optimization after all flows are assigned. Thus, the heuristic algorithm has three phases: 1) initial placement and assignment (IPA), 2) VM and request flow exchange (VRE), and 3) dead-flow re-accommodation (DR), described in Algorithms I, II, and III, respectively.

In IPA, input parameters are the same as in MILP, as stated in Section IV.B. First, IPA sorts all given flows according to their latency thresholds. Since flows with lower latency threshold have less chance to be served by remote DCs, IPA places VMs for them with higher priority to reduce their probability of being blocked. Second, among the flows that are associated to the same app and have the same priority, IPA handles them in decreasing order of the workload of each flow.

For each pending flow with the highest priority, IPA continues trying to place VMs at the nearest-available DC (measured by propagation latency) and assigns the workload of the pending flow to it, until an available DC is found. The criteria for checking availability of a DC is whether the target DC can provide sufficient hardware resources to place enough VMs for serving given workload within corresponding latency threshold. The accurate number of required VMs for supporting a given flow can be calculated using Eqns. (2), (3), and (4).

However, the DC being checked may be unable to provide sufficient hardware resources to accommodate all the workloads of a given flow, but it might be able to provide a part of the required resources. In such case, IPA will separate a given workload into sub-flows. According to C_a and the amount of available hardware resources in target DC, we know the number of VMs that the target DC can support. Then, according to Eqns. (2), (3), and (4), we can know the workload that can be hosted by the newly-placed VMs. Remaining part of the pending flow will be taken as a new sub-flow for further assignment. Note that some original flows or sub-flows may not be accommodated by any DCs; this means that such flows will be blocked. IPA marks such flows as *dead flows*, and our DR algorithm will try to take care of them.

The IPA's complexity (based on merge-sort) is $O(|A| \cdot \log(|A|)) + O(|A| \cdot (|F| \cdot \log(|F|) + |F|) \cdot (|V| \cdot \log(|V|) + |V|))$, where $|A|$ is the number of apps, $|F|$ is the number of flows of each app, and $|V|$ is the number of DCs. The maximum number of un-assigned flows of a given app is $|V|$, meaning each DC has originating requests for the given app. By dropping the low-complexity part and replacing $|F|$ with $|V|$, the complexity can further be consolidated as $O(|A| \cdot \log(|A|)) + O(|A| \cdot (|V| \cdot \log(|V|)) \cdot (|V| \cdot \log(|V|)))$. Therefore, IPA's worst case complexity is $O(|A| \cdot (\log(|A|) + |V|^2 \cdot \log^2(|V|)))$.

As already mentioned, sorting order has a strong impact on the solution quality. Fig. 3 shows an example with three edge DCs and two pending flows. If Flow-A is handled first, IPA will assign it to its nearest DC, which is the local DC Edge-1, and allocate two units of hardware (HW) to support its related VM(s). Now, Edge-1 is fully occupied, and the nearest DC for Flow-B becomes Edge-3. Flow-B, from Edge-2, has to experience 20ms network latency when traveling to Edge-3. It needs more hardware resources at Edge-3 to process faster to counteract the higher network latency, e.g., assume it needs 4 units of hardware resources at Edge-3, and total hardware consumption for Flow-A and Flow-B is 6 units.

Fig. 4 considers that Flow-A and Flow-B are handled in reverse order, which leads to a solution with lower cost. Flow-B is assigned first to Edge-1, which is 10 ms away from Edge-2, and it needs 2 units of hardware resources. Lacking local resources, Flow-A is assigned to its nearest DC Edge-3, which will introduce 5 ms network latency, and it needs 3 units hardware resource at Edge-3. Now, the total hardware consumption is 5 units, which is lower than that in Fig. 3.

Algorithm I: initial placement and assignment (IPA)

Input: see Section IV.B
1: Sort all apps in increasing order of their latency threshold
2: **for each** $a \in A$
3: Sort flows of app *a,* in decreasing order of their workload
4: **for each** flow f_s^a, which originates from edge $s \in V$
5: Sort all DCs in increasing order of their net latency to *s*
6: **for each** DC $v \in V$
7: **if** net latency $t_{s,v}$ is less than latency threshold, **then**
8: place VMs at *v* and assign all or parts f_s^a to VMs
9: **else, break**
10: **end for**
11: **if** f_s^a cannot be assigned to any DC, **then**
12: mark f_s^a as a dead flow
13: **end if**
14: **end for**
15: **end for**

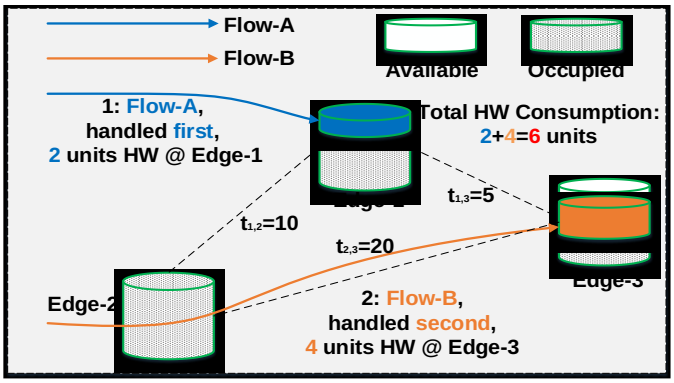

Fig. 3. Example of a non-efficient VM placement and workload assignment.

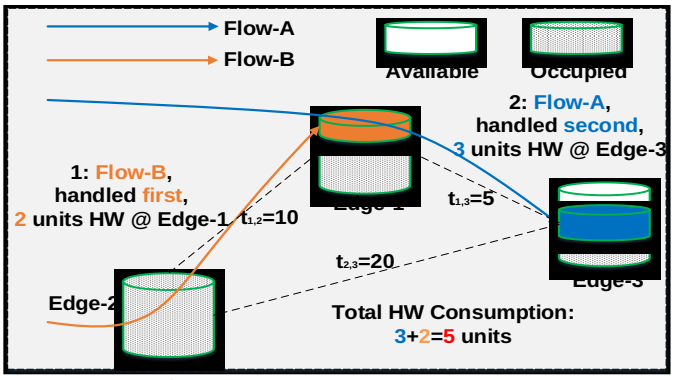

Fig. 4. Example of optimized VM placement and workload assignment.

Motivated by this observation, we design the phase-2 algorithm, called VM and request flow exchange (VRE), to improve the solution provided by IPA. VRE takes the output of IPA as its input, which is formed by a set of assigned flows and a set of dead flows. Similar to illustrations in Figures. 3 and 4, the exchange happens between two flows, and the main task of VRE is to find good candidates for exchange. An important criterion for such pairs is that total hardware consumption of

two target flows should decrease after exchanging.

Note that VRE only focuses on the exchange of big flows, as small flows' impacts on hardware consumption is relatively lower. Let F_n denote a set of big flows currently assigned to DC n. For each flow $b_{m,n}^{a1} \in F_n$, corresponding to app $a1$, originating from DC m, and assigned to DC n, we first check if there are DCs that are closer to source m than current destination n, and denote them as a set Q. If Q is not empty, it means that DCs in Q were fully occupied by other flows. Thus, for each DC $q \in Q$, we consider all flows which are currently assigned to q as potential pair flows for $b_{m,n}^{a1}$, and check whether they meet the criteria for exchanging. Let $b_{p,q}^{a2}$ denote a flow, for app $a2$, originating from DC p, and assigned to DC q. Now, we have two flows b_m^{a1} and b_p^{a2}, originated from DCs m and p respectively, and we also have two candidate destination DCs n and q. Before exchange, hardware consumptions at DCs n and q for the two flows are presented as $\mathrm{HW}(b_{m,n}^{a1})$ and $\mathrm{HW}(b_{p,q}^{a2})$; after exchange, hardware consumptions at DCs q and n for the two flows are presented as $\mathrm{HW}(b_{m,q}^{a1})$ and $\mathrm{HW}(b_{p,n}^{a2})$. Note that $\mathrm{HW}(b_{s,d}^{a})$ is calculated as $\mathrm{HW}(b_{s,d}^{a}) = C_a m_{s,d}^{a}$. If $\mathrm{HW}(b_{m,q}^{a1}) + \mathrm{HW}(b_{p,n}^{a2})$ is less than $\mathrm{HW}(b_{m,n}^{a1}) + \mathrm{HW}(b_{p,q}^{a2})$, it means that flows $b_{m,n}^{a1}$ and $b_{p,q}^{a2}$ are eligible for exchange; otherwise, VRE continues to check other potential flows. If Q is empty, VRE continues to look for other pairs for other big flows in F_n.

The steps for exchanging two eligible flows are illustrated in Procedure I. Note that two candidate flows in a pair might be unable to be exchanged entirely, depending on how much hardware resources the two destination DCs can provide for each flow. We have already mentioned that DC q is a closer DC for flow b_m^{a1}, but it is occupied by flow b_p^{a2} before assigning b_m^{a1}. It means DC q (e.g., Edge-1 in Fig. 3) was a better choice than DC n for both flows. Here VRE tries to make space for b_m^a on DC q by migrating b_p^{a2} to DC n. If $\mathrm{HW}(b_{p,q}^{a2})$, i.e., amount of released hardware resources from b_p^{a2} at DC q is larger than $HW(b_{m,q}^{a1})$, i.e., amount of required hardware resources by b_m^{a1} at DC q, part of b_p^{a2} will be migrated to original destination DC n of b_m^{a1}, and entire b_m^{a1} will be migrated to DC q. Note that remaining part of b_p^{a2} will be kept at DC q as a new sub-flow, as DC q is still a better choice for this partial flow. If $\mathrm{HW}(b_{p,q}^{a2})$ is less than $\mathrm{HW}(b_{m,q}^{a1})$, entire b_p^{a2} will be migrated to DC n to release all its occupied resources for b_m^{a1}, only part of b_m^{a1} can be migrated to DC q. If $\mathrm{HW}(b_{m,q}^{a1})$ equals to $\mathrm{HW}(b_{p,q}^{a2})$, b_m^{a1} and b_p^{a2} can be exchanged entirely. Note that according to the definition of $\mathrm{HW}(b_{s,d}^{a})$, hardware consumption of a flow at a DC is linearly-correlated with the flow's workload, and thus partial exchange can still save part of hardware resources, while keeping the latency constraint maintained. Also note that VRE is a best-effort approach with no guarantee for optimal output. However, it can be executed iteratively to improve the solution.

The complexity of VRE is calculated as $O(|F1| \cdot |V| \cdot |F2|)$, where $|F1|$ is the number of assigned flows, $|V|$ is the number of DCs, and $|F2|$ is the number of flows assigned to each DC. In the worst case, each flow is divided into separate sub-flows towards each DC. Accordingly, $|F1|$ will be $|A| \cdot |V|^2$ and $|F2|$ will be $|A| \cdot |V|$, where $|A|$ is the number of apps. Thus, the overall complexity of VRE would be $O(|A|^2 \cdot |V|^4)$. On the other hand, in practical settings, a flow will most likely be assigned to only a few (say X at max) nearby DCs and each DC can only take the flows from a few nearby DCs (say Y at max) due to the latency constraint. In that case, the complexity of $|F1|$ and $|F2|$ can be evaluated as $|A| \cdot (|V| \cdot X)$ and $|A| \cdot (Y)$, respectively, in which X and Y correspond to O(1) complexity. Therefore, the complexity can be reduced to $O(|A|^2 \cdot |V|^2)$. However, note that such relaxation will be invalid when APPs latency threshold is too loose, and remote DCs cannot be excluded from the set of candidate DCs.

Algorithm II: VM and request flow exchange (VRE)

Input: assigned big flows $\{b_{m,n}^{a1}\}$
1: **for each** flow $b_{m,n}^{a1}$
2: calculate $\mathrm{HW}(b_{m,n}^{a1})$ it needs at destination DC n
3: **for each** DC node $q \in Q$, which are closer to n than m
4: estimate $\mathrm{HW}(b_{m,q}^{a1})$ it needs at DC q
5: **if** $\mathrm{HW}(b_{m,q}^{a1}) < \mathrm{HW}(b_{m,n}^{a1})$, **then**
6: find all flows $\{b_{p,q}^{a2}\}$, which were assigned to q
7: **for each** $b_{p,q}^{a2}$, which originates from DC p to q
8: **if** q is reachable for $b_{p,q}^{a2}$, **then**
9: calculate $\mathrm{HW}(b_{p,q}^{a2})$ it required at DC q
10: estimate $\mathrm{HW}(b_{p,n}^{a2})$ if migrate it to DC n
11: **if** $\mathrm{HW}(b_{m,q}^{a1}) + \mathrm{HW}(b_{p,n}^{a2}) < \mathrm{HW}(b_{m,n}^{a1}) + \mathrm{HW}(b_{p,q}^{a2})$,
12: exchange $b_{m,n}^{a1}$ with $b_{p,q}^{a2}$(see Procedure I)
13: **end if**
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18:**end for**

Procedure I: Flow Exchange Procedure

Input: exchange-eligible flows $b_{m,n}^{a1}$ and $b_{p,q}^{a2}$, hardware consumption $\mathrm{HW}(b_{m,n}^{a1})$, $\mathrm{HW}(b_{m,q}^{a1})$, $\mathrm{HW}(b_{p,q}^{a2})$ and $\mathrm{HW}(b_{p,n}^{a2})$
1: **if** $\mathrm{HW}(b_{m,n}^{a1}) \geq \mathrm{HW}(b_{p,n}^{a2})$ *and* $\mathrm{HW}(b_{p,q}^{a2}) \geq \mathrm{HW}(b_{m,q}^{a1})$, **then**
2: release $b_{m,n}^{a1}$ from DC n and release $b_{p,q}^{a2}$ from DC q
3: assign origin flow $b_{p,q}^{a2}$ to DC n as $b_{p,n}^{a2}$
4: assign origin flow $b_{m,n}^{a1}$ to DC q as $b_{m,q}^{a1}$
5: **else if** $\mathrm{HW}(b_{m,n}^{a1}) < \mathrm{HW}(b_{p,n}^{a2})$, **then**
6: release $b_{m,n}^{a1}$ from DC n
7: assign as much $b_{p,q}^{a2}$ as DC n can host to n as $b_{p,n}^{a2}$
8: assign origin flow $b_{m,n}^{a1}$ to DC q as $b_{m,q}^{a1}$
9: **else if** $\mathrm{HW}(b_{p,q}^{a2}) < \mathrm{HW}(b_{m,q}^{a1})$, **then**
10: release $b_{p,q}^{a2}$ from DC q
11: assign as much $b_{m,n}^{a1}$as DC q can host to q as $b_{m,q}^{a1}$
12: assign origin flow $b_{p,q}^{a2}$ to DC n as $b_{p,n}^{a2}$
13:**else**, **then** error
14:**end**

VRE is designed to exchange flows which are already successfully assigned by IPA. But dead flows blocked by IPA are still blocked after running VRE. To find more chances for

accommodating such blocked flows, we design another algorithm called dead-flow re-accommodation (DR).

Taking blocked flows from IPA as input, DR accommodates blocked flows by migrating some assigned flows and making space for blocked flows. For each blocked flow f_s^{a1}, originated from DC *s,* for app $a1$, DR first sorts all DCs in increasing order of their network latency to DC *s* so that it can try to find space for f_s^{a1} from the nearest DC. For each candidate DC *v,* DR handles each flow $b_{u,v}^{a2}$, which is currently assigned to *v,* and checks whether there is a chance to migrate $b_{u,v}^{a2}$ to other DCs, which have free hardware resources. When such a flow is found, DR will calculate the hardware capacity $\mathrm{HW}(b_{u,v}^{a2})$ that can be released on DC *v* by migrating such a flow. Let $\mathrm{HW}(b_{s,v}^{a1})$ denote the required resources for hosting f_s^{a1} at DC *v.* According to the magnitude relation between $\mathrm{HW}(b_{u,v}^{a2})$ and $\mathrm{HW}(b_{s,v}^{a1})$, whole or part of $b_{u,v}^{a2}$ will be migrated from DC *v* to other available DCs to make space for the whole or part of $b_{s,v}^{a1}$ at DC *v*. There is no guarantee that DR can accommodate every dead flow successfully, and some dead flows may still be blocked because DR is a best-effort approach and it is not designed to find a feasible solution by enumerating all the possible solutions.

The complexity of DR is similar with VRE, but the maximum number of input flows is $|\mathrm{A}| \cdot |\mathrm{V}|$, meaning there are dead flows for each app at each DC. The scale of a flow is $|\mathrm{A}| \cdot (|\mathrm{V}| + \mathrm{X})$, and thus the complexity of DR is $\mathrm{O}(|\mathrm{A}|^2 \cdot |\mathrm{V}|^2)$.

Algorithm III: dead-flow re-accommodation (DR)

Input: set of blocked flows $\{f_s^{a1}\}$, which is from DC s.
1: **for each** flow f_s^{a1}
2: sort all DCs in increasing order of their net latency to s
3: **for each** DC node $v \in V$
4: **if** $t_{s,v}$ is less than latency threshold, **then**
5: find all flows $\{b_{u,v}^{a2}\}$, which were assigned to *v*
6: **for each** $b_{u,v}^{a2}$, which is from DC *u* for app *b*
7: find all DCs, which have free hardware resources and reachable for app *b* from DC *u,* as a set $\{w\}$
8: **for each** DC *w*
9: calculate $\mathrm{HW}(b_{u,w}^{a2})$, $\mathrm{HW}(b_{u,v}^{a2})$, $\mathrm{HW}(b_{s,v}^{a1})$
10: migrate $b_{u,v}^{b}$ to *w*, and assign f_s^{a} to *v*
11: **end for**
12: **end for**
13: **end if**
14: **end for**
15: **end for**

VI. Illustrative Numerical Examples

In this section, we perform a preliminary investigation of the performance of our VM placement and workload assignment methods. To get realistic data as input parameters, we choose seven cities from south-west United States as edge sites, which host edge DCs, and one city from northern-west USA as the cloud DC site. The amount of hardware resources of an edge DC is set proportionally to the city's population, and the hardware resources of a cloud DC is set to a very large value that is not expected to be exhausted. Details of edge sites and cloud node capacities are listed in Table I. Network latencies for each node pair are obtained from website [35], and they are used as inter-DC network latency.

TABLE I. DC Parameters.

City	HW resources (unit)	DC Type
San Francisco	225	Edge
Los Angeles	1000	Edge
Phoenix	375	Edge
Sacramento	125	Edge
San Jose	250	Edge
Las Vegas	150	Edge
San Diego	350	Edge
Portland	10000	Cloud

TABLE II. Applications Parameters [36].

APP (a)	T_a (ms)	u_a
Self-Driving	1	1500
Augmented Reality	2	800
Healthcare	5	500
Accelerated Video	10	400
Virtual Reality	20	300
Web Game	30	200
HD Broadcast	50	100

We consider seven sample apps, with latency requirements [36] listed in Table II. In real world, VM hardware capacities can be heterogeneous. In our simulation study, without loss of generality, we assume each VM is equipped with one unit of hardware resource and corresponding VM service capability (i.e., u_a, number of requests one VM can serve per second) that one unit of hardware resource can provide.

The population of each city varies roughly in the range [500,000, 4,000,000]. According to statistics of some sample applications (e.g., Steam), the ratio of peak concurrent users over the registered users is around 1%. Therefore, we roughly generate the overall workload of each city (city workload) proportionally to the 1% of their population, i.e., within a range of [5000, 40000]. We divided each city workload into seven parts (one for each app). In details, for each city, we randomly generate a weight within [1, 5] for each app to denote the share of workload for corresponding apps. With the weight for each app, we can divide the overall workload for each app by selecting the corresponding share for each app.

We define three parameters: workload ratio (WR), hardware capacity ratio (HR) and scale down factor (SDF), to apply sensitivity to the base settings. WR, HR, and SDF are actually the key factors for VM placement and workload assignment in the resource-constraint context, and we can test our approaches in many cases (especially the congestion cases) by varying the relationship between workload/hardware-capacity/latency. Multiplied by WR or HR, the base settings for workload or DC capacity can be zoomed in/out linearly, and we study their impacts on hardware consumption (calculated by the function in Eqn. 5) in Figures. 5-7. Dividing the inter-DC latencies by SDF, we can shrink the geographical scale of the cooperative edge nodes, and test our approaches' sensitivity to geographical scales (see Fig. 8). Note that WR, HR and SDF affect the VM placement and workload assignment decisions jointly. There are many combinations of WR, HR and SDF, and results show that WR=3, HR=1 and SDF=1 are good choices as base settings, because fixing two of them in the sensitivity test can limit the

other one in a practice range, which makes the simulation represents a real-world system behavior.

More specifically, results for the MILP model are obtained from IBM ILOG CPLEX Optimization Studio 12.6, while other results are from our lab-grown Java-based simulator. Note that in our resource planning problem, the input workload is the number of requests per second, instead of a set of requests that follow an arrival rate, and thus there is no randomness in the input and the output from the heuristics is deterministic. Therefore, there is no need to run multiple instances with the same settings to get the average value and confidence interval.

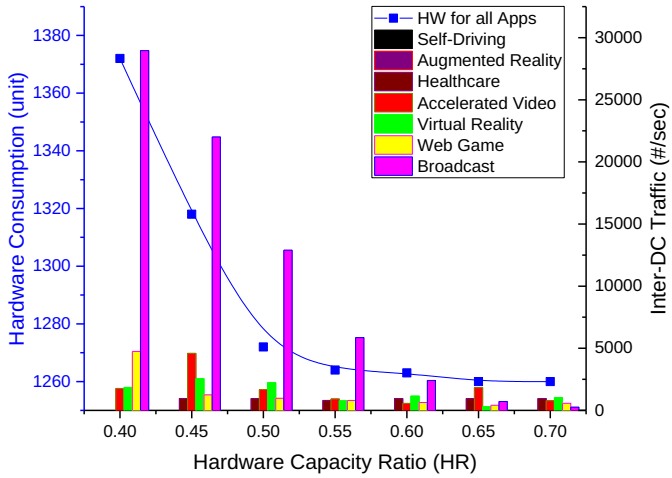

Fig. 5. Hardware consumption and inter-DC traffic for different HRs.

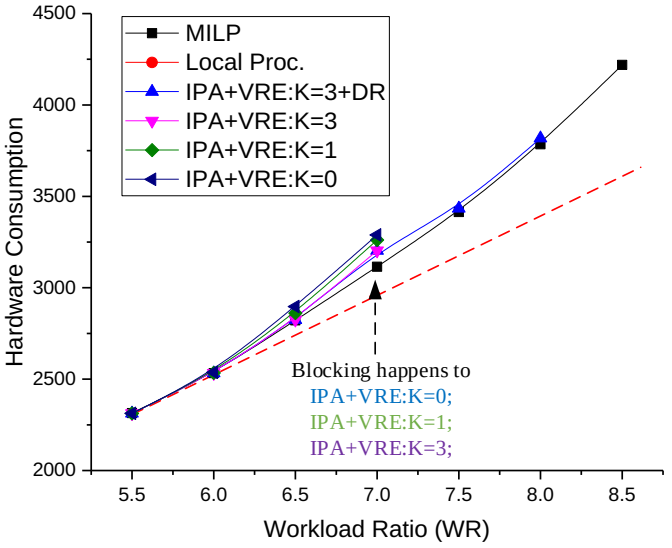

Fig. 6. Hardware consumption of different approaches vs. WR.

Fig. 5 shows the hardware consumption (on left y-axis) and inter-DC traffic (on right y-axis) for varying HR on the x-axis (fixing WR =3 and SDF=1). As expected, hardware consumption and inter-DC traffic increase when HR decreases. This is mainly due to the two following reasons. First, less hardware capacity in DCs means that congestion is achieved more quickly, and hence some requests have to go to remote DCs. Second, such inter-DC re-direction introduces higher network latency, and leaves less time for processing at destination VMs. In other words, network congestion also induces more hardware consumption as more hardware computing resources are required to process the workload faster, so that the end-to-end latency can be satisfied.

Among the seven sample apps, Fig. 5 also shows that inter-DC traffic of the least latency-sensitive app (video broadcast) is much higher than other apps. In fact, when local DCs cannot handle all the requests, the apps with looser latency requirements will be more likely to be assigned to remote DCs to make room for latency-sensitive apps at local DC.

Figs. 6 and 7 show the performance of the proposed model and heuristic in terms of hardware consumption for increasing WR and HR. Note that the point at which lines are terminated marks the last WR/HR for which all workloads can be accommodated successfully. Latency-violation or blocking arise for the next value of WR/HR. We also plot the hardware consumption of a baseline - Local Processing (Local Proc.), in which cross-DC workload assignment is disabled, to investigate the benefits of cross-DC cooperation in edge-cloud. Also note that blocking happens to baseline at lower WR and higher HR, but we plot the theoretical (assuming each DC has sufficient hardware capacity, as in cloud) hardware consumption value in dotted line, to show how congestion impacts the hardware consumption in cooperative edge-cloud.

Fig. 6 compares the hardware consumption of the proposed algorithms (with different K, the number of iterations of VRE) vs. the results obtained with the MILP, under different WR (fixing HR=1 and SDF=1). Comparing the various IPA+VRE strategies, we observe that hardware consumption under different K (i.e., iterations of VRE) is different, especially when workload ratio is high. The higher is K, the lower hardware consumption can be achieved. When K is 3, output of IPA+VRE is very close to output of MILP. We also observe that blocking happens at the same load ratio, which is WR=7, for different K. Higher K cannot reduce blocking because VRE is not capable of re-accommodating blocked flows, hence the amount of blocked requests is independent of K. If we add DR to “IPA+VRE: K=3”, the algorithm can re-accommodate some of the flows blocked in IPA. We run DR twice, and it can be noted that blocking now occurs at a larger WR (i.e., WR = 8), as DR has accommodated blocked flows at WR=7 and 7.5. We also observe that the hardware consumption of IPA+VRE+DR is very close to the MILP output. Comparing the output of heuristics and baseline in Fig.6, we see that blocking happens to Local Proc. at WR=4.5, which is much lower. This indicates that the proposed approaches can accommodate more workload without latency-violation by employing cross-DC cooperation. Further, when compare the output of heuristics with theoretical hardware consumption of Local Proc., we also observe that the hardware consumption of the heuristics increases non-linearly after Local Proc. being blocked (at WR=4 in Fig. 6 and HR=0.7 in Fig. 7, not shown), meaning cross-DC cooperation consumes extra hardware in case of congestion.

Fig. 7 compares the hardware consumption of the proposed algorithms with MILP for different HR (fixing WR=3 and SDF=1). When HR decreases, i.e., hardware capacity becomes a limitation, hardware consumptions of IPA+VRE with different K become different. Similar to Fig. 6, a higher K can

reduce hardware consumption. When HR goes down to 0.45, blocking happens to IPA+VRE, but, by running DR, blocking can be postponed to HR=0.4. Also, the hardware consumption of "IPA+VRE:K=3+DR" is quite close to the output of MILP under lower HR. Figs. 6 and 7 show that the solutions found by our algorithms are very close to the MILP solution, and they are acceptable. Comparing the output of heuristics and baseline, we can get the similar conclusion (with Fig. 6) that the cross-DC cooperation can accommodate more workloads with the cost of extra hardware consumption in case of congestion.

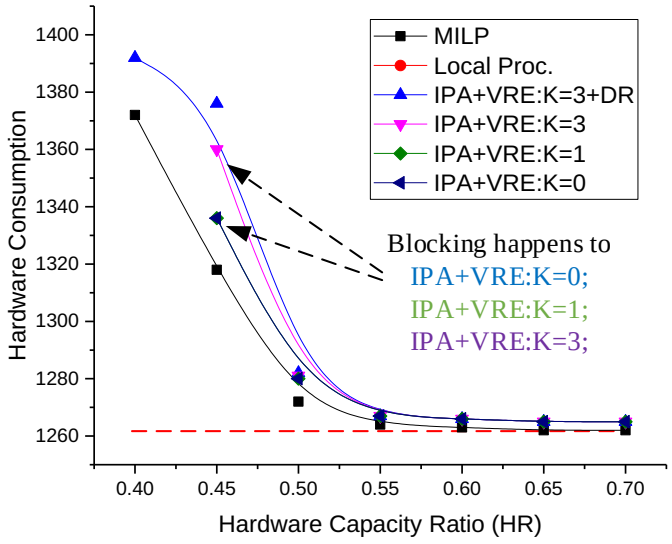

Fig. 7. Hardware consumption of different approaches vs. HR.

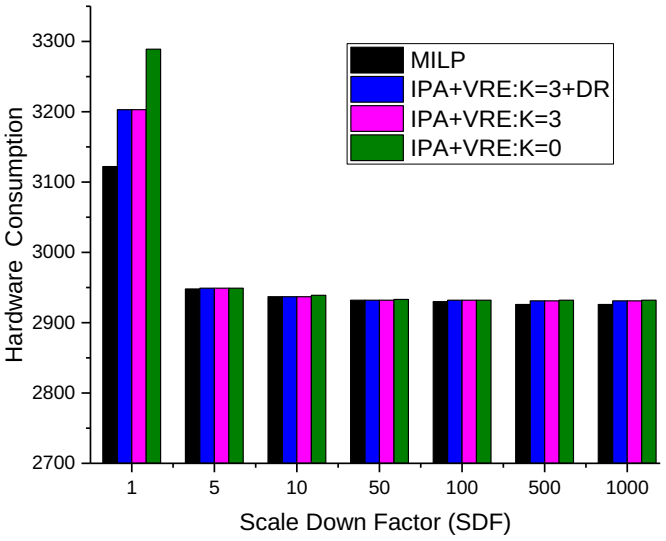

Fig. 8. Hardware consumption of different approaches vs. SDF.

Fig. 8 compares the hardware consumption of the proposed algorithms with MILP under different SDF (fixing WR=7 and HR=1). With the base settings (SDF=1), we see that IPA, IPA+VRE and IPA+VRE+DR can accommodate the given workload with a slightly higher hardware consumption than that of the MILP. When shrinking the geographical scale from 5 to 1000 times, there are two important observations. First, the overall hardware consumption for the same workload decreases when SDF increases. This is because the inter-DC latency is shortened in the shrunken topology, and thus destination DC can serve the cross-DC traffic a bit slower, with less hardware. Second, for SDF that is larger than 1, the hardware consumption from proposed heuristics is very close to that from the MILP model, indicating that the proposed heuristics are applicable to edge-cloud systems in different geographical scales. Note that we do not need to test the SDF<1 cases, because the base setting is obtained from the real world, and enlarging the base setting with SDF<1 will result in an impractical geographical scale that does not match with the low-latency edge computing context.

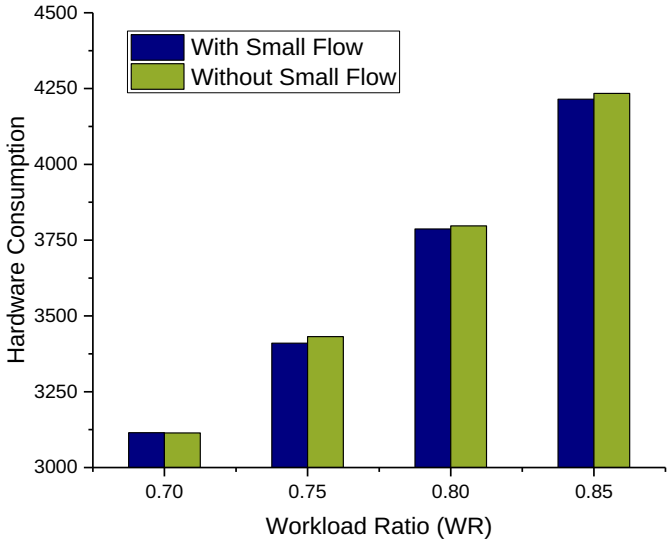

Fig. 9. Hardware consumption with/without small flow in MILP.

TABLE III. RUNNING TIME (ms) FOR PROPOSED APPROACHES WITH DIFFERENT WR SETTINGS.

WR	MILP (0.1)	MILP (0.01)	MILP (0.001)	Heuristic (8)	Heuristic (64)
4	622	954	*	5	57
4.5	1972	1318	*	5	52
5	1406	1012	*	5	55
5.5	10716	86007	*	6	55
6	60520	36913	*	6	56
6.5	45886	5275	*	6	57
7	31495	2740	*	5	54
7.5	22502	30007	*	6	52
8	25206	19103	*	7	55

In Section III, the concept of small flow was introduced as an optional choice for workload assignment process. Accordingly, we investigate the solution quality with and without small flow in our MILP model. Figure 9 compares the hardware consumption of the models with/without small flow, under relatively higher workloads. It can be noted that the model with small flow allows us to save some hardware resources. We did not plot the hardware consumption under lower WR as difference in hardware consumption become very small in this case. We conclude that small flow is helpful in improving the overall solution particularly at high workloads (indeed, in those situations when our proposed approach is more useful), but the improvement is limited to about 0.5%.

To verify the complexity of proposed heuristics and MILP model, we also compared their average running times (in ms). The running time in this table is the average value of five

executions with random workload distributions for different apps. The first column in Table III marks the given WR, while 1) columns 2-4 show the running time of MILP model with different tolerated optimality-gap (the value in brackets after MILP in first row) for 8 nodes system; and 2) columns 5-6 show the running time of heuristics for the 8 and 64 nodes systems. Note that the MILP solver may did not reach a feasible solution within 10 min for some settings, and we removed such samples from calculating the average running time. Note also that the * symbol in table III means the MILP solver did not reach a solution that satisfies the tolerated optimality-gap for any of the five settings, within 10 minutes. Comparing columns 2-4 and 5, we see the heuristics (even for larger problem) are much faster than MILP. The running time in table III was collected from an AWS c5.2xlarge EC2, with 8 vCPU, 16G Memory, and Ubuntu 18.04 operating system.

Discussion. In summary, in edge-cloud computing, where edge DCs can cooperate with each other and with cloud, the proposed MILP model and heuristic algorithms can optimize edge DCs' hardware resource efficiency from an overall perspective by leveraging cross-site cooperation. When some edge DCs are congested and cannot host all local flows, some flows will be routed to remote DCs, and corresponding VMs will be placed there. Cross-site VM placement and workload assignment can avoid blocking flows at congested DCs, but will cause some side effects. First, remote VM placement and flow assignment will introduce extra network traffic between source and destination DCs. Second, inter-DC data transmission incurs longer network latency. Since overall service latency threshold of each request is pre-set by each app, the more time spent on the network, the less time is left for queuing and processing at destination DC, and more hardware is needed to process faster. That is why the hardware consumption increases non-linearly when there are more workloads or less hardware capacities.

VII. Conclusion

This work studied the problem of virtual machine (VM) placement and workload assignment in cooperative edge-cloud computing over backhaul networks. To improve the information technology (IT) infrastructure efficiency, a mixed integer linear programming (MILP) model and a three-phase heuristic were proposed to efficiently place service VMs and assign workloads. The proposed model and algorithm were designed to reduce the overall hardware consumption for placing VMs, while meeting application's heterogeneous latency requirements. In the preliminary scenarios evaluated, the heuristics were able to reach acceptable and efficient solutions for the large-scale problems. These numerical results indicated that higher workloads and less hardware resources introduce more inter-datacenter traffic and cause extra hardware consumption; and datacenter hardware efficiency can be optimized to accommodate more workloads by leveraging the network-backed remote VM placement and cross-site cooperation. Moreover, this work also evaluated the concept of small-flow's impact on the quality of solutions, and results showed that small flow can help to save a certain amount of hardware consumption particularly in the congestion case. This work aims to provide offline solutions for initial/ incremental applications/VMs deployment, according to the estimated workloads from each edge site. In practice, the workload of each flow is not completely fixed; instead, it changes dynamically. The online solutions for hardware-efficient VM and workload management in cooperative edge-cloud over backhaul networks is another open problem.

Acknowledgement

This work has been sponsored in part by National Natural Science Foundation of China (NSFC 61822105, 61901053), the Fundamental Research Funds for the Central Universities (2019XD-A05), State Key Lab of Information Photonics and Optical Communications of China (IPOC2019ZR01), and US-Japan JUNO2 project: (NSF Grant no. 1818972).

References

[1] Y. Hu et al., "Mobile edge computing—A key technology towards 5G," ETSI White Paper (2015).
[2] Y. Mao et al., "A Survey on Mobile Edge Computing: The Communication Perspective," IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, 2017.
[3] G. Brown. "Mobile Edge Computing Use Cases & Deployment Options," Juniper White Paper (2016).
[4] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1628-1656, 2017.
[5] L. Gupta et al., "Mobile Edge Computing – An Important Ingredient of 5G Networks," IEEE Software Defined Networks Newsletter, 2016.
[6] T. X. Tran et al., "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," IEEE Communications Magazine, vol. 55, no. 4, pp. 54-61, April 2017.
[7] Singh, Kiran Deep, "Role of optical network in cloud/fog computing." Telecommunication Systems-Principles and Applications of Wireless-Optical Technologies. IntechOpen, 2019.
[8] K. Guo et al., "Joint Computation Offloading and Bandwidth Assignment in Cloud-Assisted Edge Computing," IEEE Transactions on Cloud Computing. doi: 10.1109/TCC.2019.2950395.
[9] T. Q. Dinh et al., "Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling," IEEE Transactions on Communications, vol. 65, no. 8, pp. 3571-3584, Aug. 2017.
[10] J. V. Joseph et al., "Dynamic Computation Offloading in Mobile-Edge-Cloud Computing Systems," IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 2019.
[11] Z. Usmani et al., "A survey of virtual machine placement techniques in a cloud data center." Procedia Computer Science, vol. 78, pp. 491-498, 2016.
[12] A. Beloglazov et al., "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing." Future Generation Computer Systems, vol. 28, no. 5, pp. 755-768, 2012.
[13] S. Rahman et al., "Dynamic Workload Migration over Optical Backbone Network to Minimize Data Center Electricity Cost," IEEE International Conference on Communications (ICC), France, 2017.
[14] Y. Wu et al, "Green Data Center Placement in Optical Cloud Networks," IEEE Transactions on Green Communications and Networking, vol. 1, no. 3, pp. 347-357, 2017.
[15] P. Kumar and R. Kumar, "Issues and Challenges of Load Balancing Techniques in Cloud Computing: A survey." ACM Computing Surveys vol. 51, no. 6, pp.120, 2019.
[16] "What Is Global Server Load Balancing?", [online] Available: https://www.nginx.com/resources/glossary/global-server-load-balancing/
[17] Y. Zhao et al., "Edge Computing and Networking: A Survey on Infrastructures and Applications." IEEE Access, vol. 7, pp. 101213-101230, 2019.
[18] M. Bouet and V. Conan, "Mobile Edge Computing Resources Optimization: A Geo-Clustering Approach," in IEEE Transactions on

Network and Service Management, vol. 15, no. 2, pp. 787-796, June 2018.
[19] "Optimizing Latency and Bandwidth for AWS Traffic", [online] Available: https://aws.amazon.com/blogs/startups/optimizing-latency-and-bandwidth-for-aws-traffic/
[20] "About Azure Edge Zone Preview", [online] Available: https://docs.microsoft.com/en-us/azure/networking/edge-zones-overview
[21] "Metro-Haul", [online] Available: https://metro-haul.eu/project/
[22] K. Katsalis et al., "SLA-Driven VM Scheduling in Mobile Edge Computing," IEEE CLOUD, San Francisco, 2016.
[23] T. Ouyang et al., "Follow Me at the Edge: Mobility-Aware Dynamic Service Placement for Mobile Edge Computing," IEEE Journal on Selected Areas in Communications, vol. 36, no. 10, pp. 2333-2345, Oct. 2018.
[24] R. Deng et al., "Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption," IEEE Internet of Things Journal, vol. 3, no. 6, pp. 1171-1181, Dec. 2016.
[25] M. Sourav et al., "Efficient cost-optimization frameworks for hybrid cloudlet placement over fiber-wireless networks." Journal of Optical Communications and Networking, vol. 11, no. 8, pp. 437-451, 2019.
[26] A. Yousefpour et al., " All one needs to know about fog computing and related edge computing paradigms: A complete survey," in Journal of Systems Architecture, vol. 98, pp. 289-330, 2019.
[27] N. Yu et al., "Collaborative Service Placement for Mobile Edge Computing Applications," IEEE GLOBECOM, Abu Dhabi, 2018.
[28] A. M. Maia et al., "Optimized Placement of Scalable IoT Services in Edge Computing," 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 189-197.
[29] O. Skarlat et al., "Towards QoS-Aware Fog Service Placement," 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, 2017, pp. 89-96, 2017.
[30] A. Yousefpour et al., "FOGPLAN: A Lightweight QoS-Aware Dynamic Fog Service Provisioning Framework," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5080-5096, June 2019, doi: 10.1109/JIOT.2019.2896311.
[31] L. Zhao et al., "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, July 2018.
[32] J. Wang et al., "Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach," in IEEE Transactions on Emerging Topics in Computing, doi: 10.1109/TETC.2019.2902661.
[33] W. Wang et al., "Virtual Machine Placement and Workload Assignment for Mobile Edge Computing," IEEE CloudNet, Prague, 2017.
[34] D. B. Oljira et al., "Analysis of Network Latency in Virtualized Environments," IEEE GLOBECOM, Washington, DC, 2016.
[35] Wondernetwork, [online] Available: https://wondernetwork.com/pings
[36] "5G White Paper," [online] Available: https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf