# Resource Request Dispatch in Standalone and Federated MEC Systems: A Matching Game Approach

Ming-Yi Lin, Li-Hsing Yen, and Hojjat Baghban
Department of Computer Science, College of Computer Science
National Chiao Tung University, Hsinchu, Taiwan.

***Abstract*—Multi-access edge computing (MEC) system consisting of geographically-distributed heterogeneous servers can provide low-latency virtualized resource to support computation offloading of smart devices. When bulk offloading requests comes to an MEC system, how to dispatch requests to servers so as to maximize divergent objectives of MEC service providers and users is challenging. The problem further involves money transfer when different MEC service providers can share resource to each other. In this paper, we address request dispatch issues in a standalone MEC and among federated MEC systems using matching game theory. We have adapted several classical matching algorithms to our problem. Simulation results show that we can serve more requests while still meeting latency constraints. For federated MEC systems, we can also have high revenue.**

***Index Terms*—Resource Allocation, Edge Computing, Matching Theory.**

## I. Introduction

Cloud service providers have already deployed ubiquitous cloud services so that users can access the cloud from anywhere on earth. A variety of cloud services have been provided to meet the demands of different kinds. For example, Infrastructure as a Service (IaaS) offers virtual machines (VMs) to users for running their own applications in the cloud.

Internet of Things (IoT) aims to provide connectivity of millions of smart devices to the Internet. When IoT devices or user equipments (UEs) do not have enough computation or storage capacity to run computation-intensive tasks like Virtual Reality (VR) and Augmented Reality (AR), offloading the tasks to cloud is a feasible solution. However, the latency between a device and a distant cloud may be too high for delay-sensitive or time-critical applications. Multi-access edge computing (MEC) [1] aims at providing low-latency cloud services by placing distributed clouds to the proximity of mobile users. Together with less than 1 ms standard latency in 5G, MEC could meet strict delay constraint of most time-sensitive applications. MEC also helps reducing backhaul traffic by saving the traffic between the cloud and the edge.

Fog computing is another technology that can provide computing service or resource to UEs or IoT devices yet meet the constraint of extra-low latency. Typical fog devices are access points, vehicles, smart phones, and wearable devices. Unlike MEC servers, fog devices may be mobile and volatile, and usually have limited resource. For this reason, fog devices may also offload their tasks to MEC servers.

In this paper, we assume that an MEC system comprises a set of connected heterogeneous yet geographically-distributed MEC servers that are operated and managed by a single edge service provider (ESP). IoT devices, UEs, or even fog devices as edge users may request VMs from ESP for computation offloading. The requests go toward edge access points (EAPs), which are wireless gateways to an MEC system where requests are dispatched to appropriate MEC servers. In general, a VM request can be served by any MEC server that has enough resource yet meets requirements specific to the request. However, when various VM requests come to an MEC system from different EAPs, how to conduct request dispatch so as to maximize both the requester's and the server's objectives is challenging. On one hand, requesters may prefer MEC servers that minimize the communication latency between the servers and the requesters to maintain a certain quality of service. On the other hand, MEC servers may prefer serving requests with less resource demands so as to maximize the total number of served requests, i.e., to minimize the request denial ratio.

The dispatch could be done by a single dispatcher (like Mobile Edge Orchestrator in ETSI's MEC framework [2]) in the MEC system. However, that dispatcher renders a single point of failure and may become a performance bottleneck. Furthermore, resource requests may come with *physical constraint* that can only be met by some servers. For example, requests may specify additional physical resource (e.g., GPU-enabled VM) or resource that is only accessible or valuable in some region (e.g., wireless spectrum or location-based context information). Therefore, we consider a distributed publish-subscribe messaging platform where MEC servers publish their availabilities and capabilities to EAPs which are subscribers. When resource requests come to an EAP, the EAP forwards requests on behalf of the requesters to best-suited MEC servers. MEC servers may reject some requests when they do not have enough residual capacity. Any request that gets rejected by an MEC server can be resent by the EAP to the next best-suited MEC server. This process repeats until no more matching can be arranged.

Different ESPs may have overlapping service coverage areas. If these ESPs are federated, they can collaboratively share resources and requests to increase their profits. However, MEC federation also adds another dimension to the dispatch problem because the dispatch process should consider resource provider's revenue as well as resource consumer's payment. Typically, resource requesters aim to maximize payoff and minimize latency while ESPs aim to maximize their revenue.

In this paper, we address the request dispatch problem for both standalone and federated MEC systems. We considered it as a *matching* problem between MEC users and servers. Unlike classical bipartite matching, which has a single objective that maximizes either the cardinality or the total weight of matched pairs, the matching problem here considers each party's preference on matching, rendering it a multi-objective optimization problem.

There is a trend of applying matching theory to network resource allocation and management problems recently. Pham et al. [3] modeled computation offloading in MEC as two matching problems, one for user's selections on MEC servers and the other for user's selections on offloading channels. In [4], the authors considered resource allocations in a three-tier IoT fog system, for which the pairing between fog nodes and service providers and also that between fog nodes and service users were both modeled as matching games. In this paper, we consider several possible matching models for the dispatch problem, including maximum-cardinality/weight bipartite matching, capacitated house allocation (CHA) [5], [6], stable marriage, and college admissions. We modify the original college admissions model to fit our problem, and modify associated algorithms such as CHA, Boston [7], and deferred-acceptance (DA), for request dispatch within an MEC. We model request dispatch across MECs as a matching game with (money) transfer, and propose using DA with transfer (DA-T) for resource requesters to interactively negotiate payments. Simulation results show that DA serves more requests than the counterparts in intra-ESP offloading. For inter-ESP request dispatch, DA-T makes more revenue for ESPs than Boston.

The rest of this paper is organized as follows. Sec. II presents our system model with problem formulation. Sec. III presents the proposed dispatch methods for intra- and inter-ESP request dispatch. Sec. IV shows simulation results and the last section concludes this work.

## II. System Model and Problem Formulation

### A. System model

We assume a federated edge ecosystem consisting of $n$ ESPs $\{p_1, p_2, \ldots, p_n\}$. Intra-ESP offloading model considers request dispatch only within an ESP while inter-ESP offloading model considers dispatching requests with payments across different ESPs. Each ESP $p_i$ owns a set of $m_i$ MEC servers $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,m_i}\}$.

Assume that there are $n_i$ resource requests denoted by set $Q_i = \{q_i^1, q_i^2, \ldots, q_i^{n_i}\}$ submitted to ESP $p_i$. Each request $q_i^k \in Q_i$ comes with two parameters: resource descriptor and the associated latency constraint. Let $V$ be the set of all VM types (called *flavors*) that are provided to MEC users. The resource descriptor is $\mathbf{m}_i^k = (m_i^{k,1}, m_i^{k,2}, \ldots, m_i^{k,|V|})$, where $m_i^{k,j}$ is the number of instances of VM Type $j$, where $1 \leq j \leq |V|$, that is requested by $q_i^k$. Let $t_{i,\max}^k$ be the latency constraint associated with $q_i^k$. Formally, each request $q_i^k \in Q_i$ is a pair $(\mathbf{m}_i^k, t_{i,\max}^k)$.

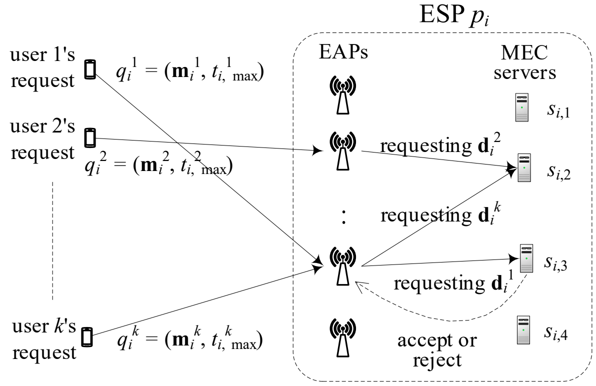
ESP $p_i$
user 1's request
$q_i^1 = (\mathbf{m}_i^1, t_{i,\max}^1)$
user 2's request
$q_i^2 = (\mathbf{m}_i^2, t_{i,\max}^2)$
user $k$'s request
$q_i^k = (\mathbf{m}_i^k, t_{i,\max}^k)$
EAPs
MEC servers
$s_{i,1}$
$s_{i,2}$
$s_{i,3}$
$s_{i,4}$
requesting $\mathbf{d}_i^2$
requesting $\mathbf{d}_i^k$
requesting $\mathbf{d}_i^1$
accept or reject

Fig. 1. An example of intra-ESP offloading with four MEC servers

Let $R$ denote the set of all different types of physical resource (CPU, memory, storage, etc.). Each server $s_{i,j} \in S_i$ has a limited supply of each resource type $r \in R$, denoted by $C_{i,j}^r$. We denote the current capacity of $s_{i,j}$ as a vector $\mathbf{C}_{i,j} = (C_{i,j}^1, C_{i,j}^2, \ldots, C_{i,j}^{|R|})$.

When MEC users need computation offloading, they send resource requests to nearby EAPs. When an EAP receives a request $q_i^k = (\mathbf{m}_i^k, t_{i,\max}^k)$, it converts $\mathbf{m}_i^k$ to a demand vector $\mathbf{d}_i^k = (d_i^{k,1}, d_i^{k,2}, \ldots, d_i^{k,|R|})$, where $d_i^{k,j}$ is the amount of the $j$-th physical resource needed by $q_i^k$. Furthermore, the EAP estimates the minimal expected latency $t_{i,j}^k$ when $q_i^k$ is served by each MEC server $s_{i,j} \in S_i$. If $t_i^k > t_{i,\max}^k$, $s_{i,j}$ cannot serve $q_i^k$ even if it has enough resource capacity. The EAP selects the best server for $q_i^k$ to send a resource request. If the request is rejected, the EAP then resends the request to the second best server, and so forth. Fig. 1 shows an example of intra-ESP offloading where $k$ requests are dispatched to four MEC servers.

For inter-ESP resource sharing, we assume that each ESP $p_i$ has an Orchestration and Control System (OCS) denoted by $OCS_i$. OCSs periodically exchange resource and latency related information following the publish-subscribe messaging paradigm. When an MEC user of $p_i$ is willing to pay for some service not provided by $p_i$, $OCS_i$ finds out all candidate MEC servers in the federated MEC systems that have enough capacity yet meet the user's latency and other constraints. For each candidate MEC server, $OCS_i$ provides information including the estimated latency and minimal selling price (i.e., the *asked price*) to the user. The user selects the most preferred server to send the request with offered price (i.e., bid). The server may accept or reject the request. If the request is not accepted, the user may in turns raise her bid and resend the request to the same server, or send the request (possibly with a different bid) to the next preferred MEC server. The process repeats until the user's request gets accepted or all candidate MEC servers reject the user's request for which the user cannot further raise the bid.

### B. Problem Formulation

We model the dispatch of resource requests to servers as a matching problem. Since requests cannot be split, $q_i^k$ for every $i$ and $k$ can only be matched to one server and served there. On the other hand, a server can be matched to several requests subject to its capacity and the latency constraints associated with requests. We use matrix $\mathbf{x}_i = [x_{i,j}^k]_j^k$ to denote the matching result in ESP $p_i$, where $x_{i,j}^k \in \{0, 1\}$ indicating whether request $q_i^k \in Q_i$ is dispatched to server $s_{i,j} \in S_i$.

Edge users and servers have different objectives. Users want to get served by servers that can minimize their latencies. Formally, the objective of every $q_i^k \in Q_i$ is

$$\max \sum_{j=1}^{m_i} \left( x_{i,j}^k \times (t_{i,\max}^k - t_{i,j}^k) \right). \tag{1}$$

On the other hand, every server $s_{i,j} \in S_i$ wants to maximize the number of requests that it can serve, i.e.,

$$\max \sum_{k=1}^{n_i} x_{i,j}^k. \tag{2}$$

The matching result is subject to several constraints. When $x_{i,j}^k \neq 0$, the amount of physical resource of type $r$ in $s_{i,j}$ that is allocated to request $q_i^k$ is $d_i^{k,r}$. The *capacity constraint* asserts that

$$\forall r \in R, \forall s_{i,j} \in S_i, \sum_{k=1}^{n_i} \left( d_i^{k,r} \cdot x_{i,j}^k \right) \leq C_{i,j}^r. \tag{3}$$

The *demand constraint* ensures that any request is either not served at all or allocated the exact amount of physical resource that it demands from a single server. More specifically,

$$\forall q_i^k \in Q_i, \forall s_{i,j} \in S_i, \sum_{j=1}^{m_i} \left( d_i^{k,r} \cdot x_{i,j}^k \right) \leq d_i^{k,r}. \tag{4}$$

and

$$\forall q_i^k \in Q_i, \forall s_{i,j} \in S_i, \; x_{i,j}^k \in \{0, 1\}. \tag{5}$$

(5) is the *no-split* constraint, which demands that no request can be split and served separately by different servers.

For request dispatch in federated MEC systems, we reuse infrastructure, resource, and capacity-related notations defined for the intra-ESP offloading problem like $S_i$, $R$, and $\mathbf{C}_{i,j}$. For request-related notations, $q_i^k$'s (and, accordingly, $Q_i$'s, $\mathbf{m}_i^k$'s, and $t_{i,\max}$'s) are submitted toward ESP $p_i$. These requests are from other ESPs $p_j \neq p_i$.

The operational cost of each server $s_{i,j}$ is defined on each resource type as $\mathbf{c}_{i,j} = (c_{i,j}^1, c_{i,j}^2, \ldots, c_{i,j}^{|R|})$, where $c_{i,j}^r$ is the unit operational cost of physical resource type $r$ at $s_{i,j}$. With $\mathbf{c}_{i,j}$, the total operational cost incurred by request $q_i^k$, if granted, can be calculated as

$$\theta_{i,j}(q_i^k) = \sum_{r=1}^{|R|} \left( c_{i,j}^r \cdot d_i^{k,r} \right). \tag{6}$$

Let $b_{i,j}^k$ be the actual payment of $q_i^k$ when it is served by $s_{i,j}$. Only if $b_{i,j}^k > \theta_{i,j}(q_i^k)$ can $s_{i,j}$ serve $q_i^k$. Each ESP $p_i$'s objective is to maximize its own profit defined as

$$\max \sum_{j=1}^{m_i} \sum_{k=1}^{n_i} x_{i,j}^k \left( b_{i,j}^k - \theta_{i,j}(q_i^k) \right). \tag{7}$$

On the other hand, the requester who submits $q_i^k$ has a valuation $v_i^k$ on the request. This serves as the highest possible payment that the requester is willing to pay to ESP $p_i$ when $q_i^k$ is granted. The requester of $q_i^k$ wants to maximize the difference between $v_i^k$ and $b_{i,j}^k$ (i.e., payoff) while minimizing the latency. We thus define the requester's utility as the payoff per unit of latency time. Formally, the objective of every $q_i^k \in Q_i$ is

$$\max \sum_{j=1}^{m_i} \left( x_{i,j}^k \cdot \frac{v_i^k - b_{i,j}^k}{t_{i,j}^k} \right). \tag{8}$$

Their objectives are still subject to the capacity, demand, and no-split constraints defined above.

## III. Proposed Mechanisms

### A. Problem Analysis

By (1), $x_{i,j}^k$ should not be 1 if $t_{i,j}^k > t_{i,\max}^k$. Therefore, we can construct a bipartite graph $G_i = (\bar{S}_i \cup \bar{Q}_i, \bar{E}_i)$ for each ESP $p_i$ such that each vertex $s_j \in \bar{S}_i$ corresponds to server $s_{i,j} \in S_i$, each vertex $q_k \in \bar{Q}_i$ represents request $q_i^k \in Q_i$, and $(s_j, q_k) \in \bar{E}_i$ if and only if $t_{i,j}^k < t_{i,\max}^k$. In this way, the request dispatch problem in $p_i$ becomes a matching problem in $G_i$.

Classical bipartite matching problem is to maximize the number of matched pairs (i.e., maximum-cardinality bipartite matching). This formulation only reflects the objective of the servers. When a request prefers some server to another because the former offers a lower latency, that preference cannot be captured by the cardinality-based objective.

We may recap the preference of all possible matched pairs by weights associated with corresponding edges in the bipartite graph. Accordingly, the objective is to maximize the total weight associated with all matched pairs (i.e., maximum-weight bipartite matching). However, this modeling does not fit when the two peers in a match may have different valuations on that match.

Gale and Shapley [8] considered marriage as a matching problem between males and females where each individual's valuation matters. In marriage, each male can be matched to only one female (and vice versa), which makes marriage a one-to-one matching problem. It has been used to model allocation of wireless channels to mobile users [3]. However, our problem is not a one-to-one matching because an MEC server can serve more than one requests (subject to the capacity constraint).

Capacitated house allocation (CHA) is a many-to-one matching problem, which is to allocate a set of houses to a bunch of agents. It extends bipartite matching in two ways. First, agents can have preference on houses. Second, every house has a capacity which specifies the maximal number of

TABLE I
DIFFERENT MATCHING PROBLEMS

| Problem | Type | Preference |
|---|---|---|
| Maximum-Cardinality Bipartite Matching | One-to-one | No |
| Maximum-Weight Bipartite Matching | One-to-one | On matches |
| Capacitated House Allocation (CHA) | Many-to-one | One-sided |
| Stable Marriage | One-to-one | Two-sided |
| College Admissions | Many-to-one | Two-sided |

agents that it can accommodate. However, CHA still does not well fit our problem due to the following reasons. First, both requesters and servers have preference in our problem, but CHA considers only one-sided preference. Second, agents are assumed to have equal size, so the maximal number of agents that can be accommodated in each house is fixed and known. In contrast, requests come with different sizes (amounts of requested resource) in our problem. Consequently, an MEC server may fulfill the aggregated demand of three requests but not that of another two.

College admissions problem [8] is also a many-to-one matching, where a college can give admissions to more than one students. It differs from CHA in that colleges have preference on applicants. In [3], computation offloading from a set of mobile users to a set of MEC servers is modeled as a many-to-one matching game with the assumption that each server can execute at most some certain number of computation tasks. Here both mobile users and MEC servers have preference for the other party, so it falls into the college admission problem. However, college admissions model still does not perfectly fit our problem as each college has a fixed and known quota for admissions. In contrast, the maximal number of requests that a server can serve is not fixed.

Table I summarizes all mentioned models of matching problem.

### B. Intra-ESP Request Dispatch

We model intra-ESP request dispatch as a many-to-one, two-sided preference matching game. For each request $q_i^k \in Q_i$, the latency when it is served by each server $s_{i,j} \in S_i$ is a vector $\mathbf{t}_i^k = (t_{i,1}^k, t_{i,2}^k, ..., t_{i,m_i}^k)$. The preference value of $q_i^k$ when it is served by $s_{i,j}$ is

$$\phi_i^k(s_{i,j}) = t_{i,\max}^k - t_{i,j}^k. \tag{9}$$

The preference list of $q_i^k$ denoted by $PQ_i^k$ contains all $s_{i,j}$'s in an ascending order of $\phi_i^k(s_{i,j})$ for which $\phi_i^k(s_{i,j}) > 0$. The EAP receiving $q_i^k$ sends resource request $\mathbf{d}_i^k$ on behalf of $q_i^k$ to the most preferred MEC server in $PQ_i^k$. If the request gets rejected, the EAP then sends the request to the second preferred MEC server in $PQ_i^k$, and so forth.

Each server, on the other hand, may receive multiple resource requests in a short time. If its residual capacity can accommodate all the requests, all requests will be accepted. Otherwise, the server must identify the best subset of requests to grant so as to maximize the number of served requests. It is a bin packing problem, which is known to be NP-hard. As a heuristic, we define a preference function $\Phi_{i,j}(q_i^k)$ for each each MEC server $s_{i,j}$ as follows:

$$\Phi_{i,j}(q_i^k) = \sum_{r=1}^{|R|} \left( w_{i,j}^r \cdot (1 - \frac{d_i^{k,r}}{C_{i,j}^r}) \right), \tag{10}$$

where $w_{i,j}^r$ is a real number in $[0, 1]$ that represents the weight (e.g., scarcity) of physical resource type $r$ to $s_{i,j}$ with the property that $\sum_r w_{i,j}^r = 1$. All requests $q_i^k$'s toward $s_{i,j}$ are granted one by one in the increasing order of their preference function values. When the server can no longer grant an request with its residual capacity, it rejects the request and all requests with lower preference function values (if any).

An MEC server may have accepted some requests but have to reject some other requests later due to insufficient residual capacity. When this happens, it is an issue whether the MEC server should retract a previous grant to make room for a new request simply because the new one has a higher preference function value than the previous[1]. If we allow retraction, it is a variant of *deferred-acceptance* (DA) algorithm [8], which possesses a property that servers may *tentatively* accept requests. If acceptance is always firm (cannot be retracted), the approach is a variant of Boston [7].

### C. Inter-ESP Request Dispatch

For inter-ESP request dispatch, each MEC server $s_{i,j}$ values all requests toward it by the following preference function $\Phi_{i,j}(q_i^k)$:

$$\Phi_{i,j}(q_i^k) = \frac{b_{i,j}^k - \theta_{i,j}(q_i^k)}{\sum_{r=1}^{|R|} \left( w_{i,j}^r \cdot d_i^{k,r} / C_{i,j}^r \right)}. \tag{11}$$

Intuitively, this function gives the profit per unit of the weighted sum of all normalized physical resource requested.

On the other hand, the requester of $q_i^k$ ($q_i^k$ for short hereafter) values each candidate MEC server $s_{i,j}$ by the following preference function:

$$\phi_i^k(s_{i,j}) = \frac{v_i^k - b_{i,j}^k}{t_{i,j}^k}, \tag{12}$$

where $b_{i,j}^k$ is the price offered by $q_i^k$ to $s_{i,j}$. This is exactly the utility of $q_i^k$ being served by $s_{i,j}$. Note that $q_i^k$ may offer different prices to different servers for the same request. The offered price depends on the minimal selling price $\theta_{i,j}(q_i^k)$ provided by $s_{i,j}$, expected latency $t_{i,j}^k$ when $q_i^k$ is served by $s_{i,j}$, and the valuation on the request $v_i^k$. Clearly, $q_i^k$ will not consider $s_{i,j}$ if $\theta_{i,j}(q_i^k) > v_i^k$. If $\theta_{i,j}(q_i^k) < v_i^k$, $q_i^k$ can have a positive utility if $\theta_{i,j}(q_i^k) \leq b_{i,j}^k < v_i^k$. To maximize its utility, $q_i^k$ tends to minimize $b_{i,j}^k$. However, as $s_{i,j}$ ranks all requests toward it by (11), a request with low offered price also has low chance to be granted. When a request is rejected, the requestor can either raise the price to $b_{i,j}^k + \delta$ and resend the request to the same server or send a different offered price to another MEC server. The decision depends on which one can give the

[1] Of course, the rejected request can sent another request to the next preferred MEC server.

TABLE II
SIMULATION SETTING FOR SERVER CAPACITY

| Parameter | Distribution | Mean | Standard Deviation |
|---|---|---|---|
| Number of CPU cores | Gaussian | 50 | 10 |
| Amount of memory (GB) | Gaussian | 500 | 50 |
| Amount of storage (GB) | Gaussian | 500 | 100 |

TABLE III
VM INSTANCE TYPES OFFERED BY AMAZON EC2–US WEST (NORTHERN CALIFORNIA) REGION

| | Medium | Large | XLarge | 2XLarge |
|---|---|---|---|---|
| CPU core | 1 | 2 | 4 | 8 |
| Memory (GB) | 3.75 | 7.5 | 15 | 30 |
| Storage (GB) | 4 | 32 | 80 | 160 |

requester a higher utility. The process repeats until all requests either get granted or cannot further raise their offered prices. This is a modification of the DA algorithm with the inclusion of monetary exchange, which is similar to [9].

## IV. SIMULATION RESULTS

We conducted a series of simulations to investigate the performance of the proposed mechanisms and compare it with that of others. Both intra-ESP and inter-ESP request dispatches were considered.

### A. Environment Settings

We randomly placed ten EAPs (numbered from 0 to 9) in a $100 \times 100$ km$^2$ area. Each EAP was co-located with an MEC server. The capacity of each server was randomly determined with the setting shown in Table II. We varied the number of requests from 50 to 1000. To generate non-uniform distributions of user requests on EAPs, the identifier of the serving EAP was set by applying a floor function to a Gaussian distributed random variable (with mean 5 and standard deviation 2.5) truncated at 0 and 9.

We assumed four types of VMs as those offered by Amazon EC2 in US West Region (Table III). Each request was a combination of these flavors with most requests demanded Medium and Large VMs. Only a few demanded XLarge and 2XLarge ones. About 60% requests had latency constraints uniformly set in the range from 1 to 100 ms. Other requests did not have latency constraints. When a request was served by the MEC server co-located with the serving EAP, the latency was assumed 1 ms. When the request was served by any other MEC server, the latency was 1 ms plus the propagation delay between the MEC server and the serving EAP. For inter-ESP resource sharing, the cost of physical resources were set based on real prices [10], [11], [12].

### B. Intra-ESP Request Dispatch

We tested several matching mechanisms, including CHA, adapted Boston, and adapted DA, and compared the results with those of random matching and no offloading (denoted by No-Share). In No-Share, requests were always dispatched to the MEC servers co-located with the respective serving EAPs of the requests.

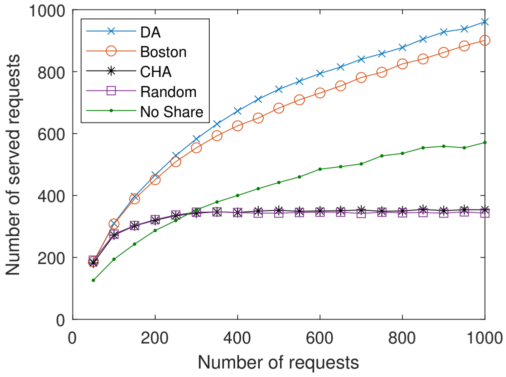

Fig. 2. Total numbers of served requests in intra-ESP request dispatch

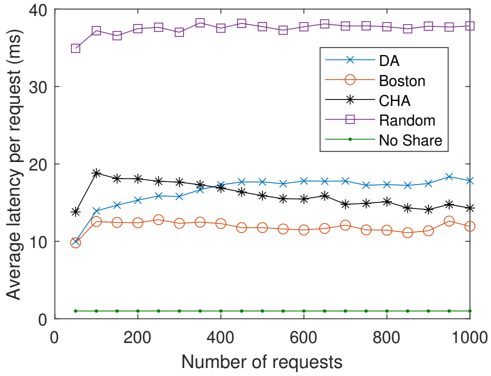

Fig. 3. Average latency per request in intra-ESP request dispatch

Figure 2 shows how the number of served requests changes with increasing number of requests using different approaches. DA clearly outperforms all others, followed by Boston. CHA and Random roughly performed the same. They performed better than No-Share only with few requests. The reason is that servers in CHA did not have preference on requests, so the set of requests that were granted when a server did not have enough capacity was not carefully determined. This is like Random.

Figure 3 shows the average latency per granted request. Here No-Share had the lowest latency, which is reasonable because only local requests could be granted. Random had the highest latency, which is also predictable. The superiority of Boston over DA comes from the property that once Boston grants a request, it never retracts the grant. Therefore, granted requests tended to be dispatched to their most preferred MEC servers. In contrast, DA may retract a request grant to make room for another request that is more preferable. Therefore, granted requests were less likely to be matched to their most preferred servers. Together with Fig. 2, we can see that this strategy is to trade requester's preference for server's.

### C. Inter-ESP Resource Sharing

For inter-ESP resource sharing, we primarily considered DA with transfer (referred to as DA-T). In DA-T, different

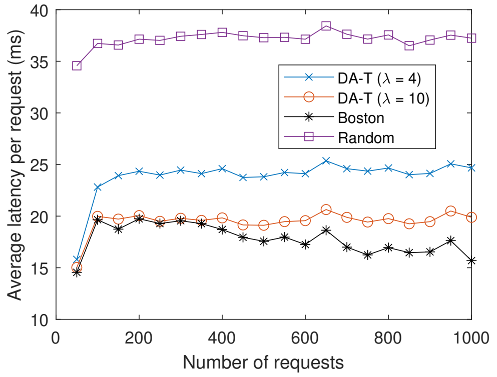

Fig. 4. Average latency per request in inter-ESP offloading

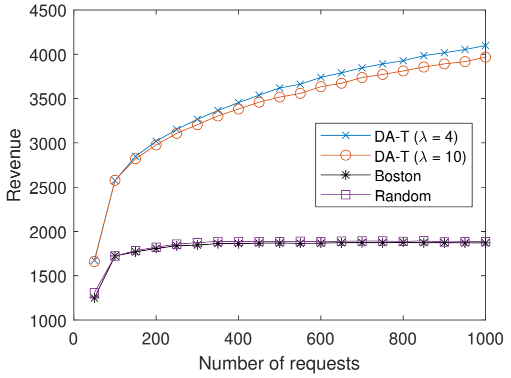

Fig. 5. Total revenue in inter-ESP offloading

requesters may have different settings on the increments of their bids (the value of $\delta$) when the proposed bids are not accepted. Generally speaking, servers prefer higher $\delta$ value while requesters prefer lower. We used a parameter $\lambda$ to set up the maximal number of times that each requester is allowed to raise its bid toward the same server. It indirectly controls the granularity of $\delta^k_{i,j}$ ($\delta$ for each $q^k_i$) as follows:

$$\delta^k_{i,j} = \frac{v^k_i - a^k_{i,j}}{\lambda}, \qquad (13)$$

where $a^k_{i,j}$ is the asked price $s_{i,j}$ provides to $q^k_i$. In our simulations, we assumed that $a^k_{i,j} = \theta_{i,j}(q^k_i)$.

Figure 4 shows the average latency per granted request in inter-ESP resource sharing. The performance of Random and Boston was expected. DA-T with $\lambda = 10$ had a lower latency than DA-T with $\lambda = 4$. This can be justified as a small granularity of bid increment ($\lambda = 10$) gave requests more chances to be considered by their most preferred servers (before switching to less preferred servers in their preference lists).

Figure 5 shows total revenue of the system. Though Boston gave granted requests low latencies, the revenue of the system was nearly the same as Random. The reason is that it did not give MEC servers the opportunity to replace low-profit requests with high-profit ones. DA-T outperformed Boston because it allows such replacements. Here large granularity of bid increment ($\lambda = 4$) gave the ESPs higher revenue, which is intuitive. However, the gap is not significant.

## V. Conclusions

We have modeled resource request dispatch in an MEC system as a many-to-one matching with two-sided preference. In this model, requests prefer MEC servers that give them short latencies while servers prefer requests that demand less resource (based on the server's own weightings on different types of resources). Approaches including CHA, Boston, and DA have been considered. For federated MEC system, resource requests come with offered payment. Requesters prefer servers that give them high payoff per unit time of latency while servers prefer requests that bring in high profit per unit of weighted, normalized physical resource. The proposed approach, DA-T, enables interactive payment negotiation. Simulation results show that DA serves more requests than the counterparts in intra-ESP offloading. For inter-ESP request dispatch, DA-T makes more revenue for ESPs than Boston.

## Acknowledgements

This work was partially supported by the Ministry of Science and Technology, Taiwan, under grant numbers 106-2221-E-009-004 and by the H2020 collaborative Europe/Taiwan research project 5G-CORAL (grant number 761586).

## References

[1] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.

[2] ETSI, "Mobile Edge Computing (MEC); terminology, v1.1.1," European Telecommunications Standards Institute (ETSI), Group Specification (GS), Mar. 2016, MEC Standard 001.

[3] Q.-V. Pham, T. LeAnh, N. H. Tran, B. J. Park, and C. S. Hong, "Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach," *IEEE Access*, vol. 6, pp. 75 868–75 885, Dec. 2018.

[4] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier IoT fog networks: A joint optimization approach combining Stackelberg game and matching," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1204–1215, Oct. 2017.

[5] D. F. Manlove and C. T. S. Sng, "Popular matchings in the capacitated house allocation problem," in *Algorithms – ESA 2006*, Y. Azar and T. Erlebach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 492–503.

[6] C. T. S. Sng, "Efficient algorithms for bipartite matching problems with preferences," Ph.D. dissertation, University of Glasgow, 2008.

[7] A. Abdulkadiroğlu and T. Sönmez, "School choice: A mechanism design approach," *American Economic Review*, vol. 93, no. 3, pp. 729–747, Jun. 2003.

[8] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[9] J. Alexander S. Kelso and V. P. Crawford, "Job matching, coalition formation, and gross substitutes," *Econometrica*, vol. 50, no. 6, pp. 1483–1504, Nov. 1982.

[10] (2018, Apr.) Price chart of amd and intel desktop cpus (latest). [Online]. Available: http://www.cpu-world.com/Price_Compare/Desktop_CPU_prices_(latest).html

[11] J. C. McCallum. (2018, Apr.) Memory prices (1957-2017). [Online]. Available: https://jcmit.net/memoryprice.htm

[12] A. Klein. (2017, Jul.) Hard drive cost per gigabyte. [Online]. Available: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/