This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

DATS: Dispersive Stable Task Scheduling in Heterogeneous Fog Networks

Zening Liu, Xiumei Yang, Yang Yang, *Fellow, IEEE*, Kunlun Wang, and Guoqiang Mao, *Fellow, IEEE*

***Abstract*—Fog computing has risen as a promising architecture for future Internet of Things (IoT), 5G and embedded artificial intelligence (AI) applications with stringent service delay requirements along the cloud to things continuum. For a typical fog network consisting of heterogeneous fog nodes (FNs) with different computing resources and communication capabilities, how to effectively schedule complex computation tasks to multiple FNs in the neighborhood to achieve minimal service delay is a fundamental challenge. To tackle this problem, a new concept named processing efficiency (PE) is first defined to incorporate computing resources and communication capacities. Further, to minimize service delay in heterogeneous fog networks, a scalable, stable and decentralized algorithm, namely dispersive stable task scheduling (DATS), is proposed and evaluated, which consists of two key components: (i) a PE-based progressive computing resources competition (PCRC) and (ii) a QoE-oriented synchronized task scheduling (STS). Theoretical proofs and simulation results show that the proposed DATS algorithm can achieve effective tradeoff between computing resources and communication capabilities, thus significantly reducing service delay in heterogeneous fog networks.**

***Index Terms*—Fog computing, task scheduling, computation offloading, matching theory.**

I. Introduction

With the rapid development of Internet of Things (IoT), 5G and embedded artificial intelligence (AI), a seamless connectivity of numerous smart devices is to be established, and a number of new applications and services, such as connected vehicles, interactive gaming and augmented reality (AR), have sprung up [1]. On one hand, the emerging various applications and the massive diverse data from devices are typically resource-hungry; on the other hand, smart devices are usually resource-constrained because of physical size constraints and energy consumption consideration [2]. To relief such tension, mobile cloud computing (MCC) has been proposed to allow end devices to offload their computation-intensive tasks to the remote resource-rich cloud via wireless connections [3]. However, due to intermittent wireless connections, scarce spectrum resources, and high propagation delay, MCC can not meet the stringent ultra-reliable and low-latency requirements of many applications, such as connected vehicles and AR.

Therefore, an emerging paradigm called fog computing has been introduced [4]. Fog computing transfers computing, storage, control and networking functions to the continuum along the cloud to things. It allows a crowd of neighbouring end-user, network edge and access devices to cooperatively accomplish resource-hungry tasks. As a result, a number of tasks originally targeting at the cloud can be effectively completed at edge by surrounding dispersive computing resources. Since fog computing carries out a considerable amount of data storage, computing and communication near end users, it enables low-latency, high-reliability, location-awareness and privacy-preservation services [5], [6].

In fog computing, how to effectively map computation tasks to a group of available heterogeneous fog nodes (FNs) is a fundamental challenge [5]. There have been some works focusing on minimizing the end-to-end service delay in fog computing [7]–[13]. Souza *et al.* [8] introduced a highly abstract hierarchical architecture called Combined Fog-Cloud (CFC) in an IoT scenario. They formulated the service allocation problem as an integer linear programming problem to minimize the latency. Yousefpour *et al.* [9] and Masri *et al.* [10] proposed two general frameworks for IoT-fog-cloud scenarios. In these two frameworks, FNs could not only execute tasks or forward tasks to the cloud, but also collaborate with each other. Through communicating with neighboring FNs, they could find a best one to execute tasks to minimize the service delay. However, both frameworks only enabled a task to be offloaded to and executed by a single FN, which could not meet the ultra-latency requirement since the computational capability of a single FN was limited. One promising solution is to execute tasks via distributed computing by multiple FNs. In [11] and [12], the authors naturally considered the scenario where a task could be splitted into fragments and offloaded to multiple FNs for distributed computing. Shih *et al.* [11] presented a framework for Fog-Radio Access Network (F-RAN) to support ultra low-latency applications. In this framework, a master F-RAN node was responsible for deciding the optimal number of participating FNs and choosing suitable ones from the candidates to join the tasks execution. Besides, the master FN was also in charge of deciding the number of allocated radio resource blocks and the number of assigned tasks for each participating FN. Lee *et al.* [12] proposed a framework to minimize the maximum computation latency under uncertainty on the arrival

Z. Liu and Y. Yang are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China, with the University of Chinese Academy of Sciences, Beijing 101408, China, and also with the Shanghai Institute of Fog Computing Technology, Shanghai 201210, China.

X. Yang and K. Wang are with the Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 200050, China, and also with the Shanghai Institute of Fog Computing Technology, Shanghai 201210, China.

G. Mao is with the School of Computing and Communication, University of Technology Sydney, Sydney, NSW 2007, Australia.

Corresponding author: Yang Yang (yang.yang@wico.sh).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

of neighboring FNs. The framework utilized the queuing theory to model the process of computation offloading and adopted the online secretary algorithm to select the optimal set of FNs. Nonetheless, both of them only considered the simple single-user scenario. Shih *et al.* [11] extended their work to the multi-user scenario in [13]. In [13], a heuristic algorithm based on dynamic programming was developed to minimize the total service latency among all users.

In addition to the latency minimization, the energy-latency tradeoff is another focus of research [14]–[21]. Pu *et al.* [14] proposed a novel mobile computation offloading framework, namely device-to-device (D2D) fogging, for fog computing, where mobile users could share the communication and computation resources with each other. They developed an online computation offloading algorithm leveraging Lyapunov optimization to minimize the time-average energy consumption of all users. They extended their work in [15] to take energy-latency tradeoff into account. In [15], devices were allowed to flexibly choose one of three among local mobile execution, D2D offloaded execution and cloud offloaded execution. The total task execution cost, including energy and latency, was minimized by transforming it to a minimum weight matching problem in a three-layer graph and applying the Edmonds's Blossom algorithm to solve it. Ti *et al.* [16] introduced a computation offloading solution exploiting computing resources from both mobile edge cloud and mobile peers. It enabled the source users, the fixed helping users and the edge cloud to collaborate with each other to execute tasks. The optimization problem was formulated into a minimum weighted energy consumption problem subject to the resources constraints and the latency requirements, and solved by a method based on the successive convex approximation and the geometric programming. Meng *et al.* [17] considered a hybrid computation offloading problem with two types of computation offloading destinations: cloud computing servers and fog computing servers, and minimized the total energy consumption under the given delay constraints. A similar problem was investigated by Deng *et al.* [18]. The authors considered the problem of minimizing the power consumption of the fog-cloud computing system while guaranteeing the required delay. They decomposed the primal problem into three subproblems and solved them via interior-point method, generalized Benders decomposition method and Hungarian method, respectively.

However, there remain many challenges and problems requiring to be addressed and analyzed in heterogeneous fog networks. For example,

- A multi-user system model where each task can be executed via distributed computing by multiple FNs is more desirable, compared with single-user scenarios [11], [12], [17], [21] or multi-user scenarios where each task can be only executed by single FN [9], [10], [14], [15].
- A solution with scalable and decentralized capabilities is much more preferred, while most works run in a centralized way with high computational complexity and heavy control signaling [13]–[21].
- The stability of solutions, an outcome where no participants want to deviate, is also an important concern, which is ignored by most of the existing works [8]–[21].

In fog computing networks, it is common that multiple end devices have the demand for computation offloading simultaneously. Furthermore, due to the limited computing capability of a single FN, it is reasonable to execute tasks via distributed computing by multiple FNs to meet the ultra-low latency requirement. Besides, with the explosion of various devices, optimal solutions, which are accompanied by global information and centralized control, will yield significant communication overhead and high computation complexity, which is a great obstacle to achieving low-latency and agile response. Finally, since devices are endowed with intelligence, they become intelligent and powerful, and meanwhile rational and selfish. Optimal solutions usually lead to an unstable outcome where some participants want to deviate.

Therefore, in this paper, we consider a general multi-user system model for a typical heterogeneous fog network, where tasks can be simultaneously offloaded to and executed by multiple dispersive FNs along with the cloud via distributed computing. We minimize the service delay of the network and develop a scalable, stable and decentralized algorithm based on the stable matching theory.

Compared with existing works, the main contributions of this paper are summarized as follows.

- A general multi-user system model for a typical heterogeneous fog network is proposed, where dispersive task nodes can simultaneously offload computation tasks to multiple neighboring helper nodes with heterogeneous capabilities and the cloud so that tasks can be executed in parallel. More importantly, a new concept named processing efficiency (PE) is defined to incorporate computing resources and communication capabilities.
- To minimize service delay in heterogeneous fog networks, a scalable, stable and decentralized algorithm called dispersive stable task scheduling (DATS) is proposed, which consists of two key components: (i) a PE-based progressive computing resources competition (PCRC) and (ii) a QoE-oriented synchronized task scheduling (STS). Particularly, to tackle the critical difficulties of constructing the preference profile in PCRC, we further develop the progressive most-preferred coalition selection (PMCS) algorithm, a key step in PCRC.
- Extensive simulations are conducted to demonstrate the performance of the proposed DATS algorithm. Theoretical proofs and simulation results show that the DATS algorithm can achieve effective tradeoff between computing resources and communication capabilities, thus significantly reducing service delay in heterogeneous fog networks.

The rest of this paper is organized as follows. We first introduce the system model and formulate our task scheduling problem in Section II. Then, we reformulate the problem as a many-to-one matching and propose the DATS algorithm in Section III. We evaluate the proposed algorithm and analyze the results in Section IV. Finally, we conclude this article in Section V.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

II. System Model and Problem Formulation

In this section, we first introduce the system model and then formulate the problem formally.

A. System Overview

We first have an overview of the system model. As illustrated in Fig. 1, we consider a general hybrid cloud-fog architecture consisting of multiple FNs and the remote cloud. The FNs can share resources and serve as helpers to each other. At every task scheduling interval, FNs with computation tasks can offload tasks to neighboring idle FNs and the cloud to reduce the latency. For convenience, we call FNs with computation tasks as task nodes. We denote the set of task nodes as $\mathcal{T} = \{1, 2, \ldots, N\}$, and use n to refer to the n-th task node. Correspondingly, we define idle FNs as helper nodes, and denote them by $\mathcal{H} = \{1, 2, \ldots, M\}$. We use m to denote the m-th helper node. Besides, the cloud consists of a set $\mathcal{V} = \{1, 2, \ldots, K\}$ of virtual machines (VMs), and we use k to signify the k-th VM.

Due to some constraints, such as the user service level, the limited hardware capability and the fairness consideration, task node n can offload computation tasks to at most q_n helper nodes and VMs, totally. On the other side, each helper node and VM can only host the computation task from a single task node at every scheduling interval (as shown in Fig. 1). Here, we assume that helper nodes and VMs are with enough storage. We represent the group of helper nodes and VMs for task node n, termed *coalition*, as $\mathcal{C}_n$ with $\mathcal{C}_n \subseteq \mathcal{H} \cup \mathcal{V}$ and $|\mathcal{C}_n| \leq q_n$. $|\mathcal{C}_n|$ is the cardinality of $\mathcal{C}_n$. In addition, we denote the coalition of helper nodes for task node n and the coalition of VMs for task node n as $\mathcal{C}_n^{\mathcal{H}}$ and $\mathcal{C}_n^{\mathcal{V}}$, respectively. $\mathcal{C}_n^{\mathcal{H}} \subseteq \mathcal{H}$, $\mathcal{C}_n^{\mathcal{V}} \subseteq \mathcal{V}$, and $\mathcal{C}_n^{\mathcal{H}} \cup \mathcal{C}_n^{\mathcal{V}} = \mathcal{C}_n$. Then, the task allocation vector can be represented as $\boldsymbol{\alpha}_n = \{\alpha_{n,l}, \alpha_{n,m}, \alpha_{n,k} \in \mathbb{R}_+ | \alpha_{n,l} + \sum_{m \in \mathcal{C}_n^{\mathcal{H}}} \alpha_{n,m} + \sum_{k \in \mathcal{C}_n^{\mathcal{V}}} \alpha_{n,k} = 1\}$, where $\alpha_{n,l}$, $\alpha_{n,m}$ and $\alpha_{n,k}$ are the fraction of tasks computed by task node n locally and the fraction of tasks offloaded from task node n to helper node m and VM k, respectively.

We introduce a parameter tuple $\{I_n, \eta_n, \mu_n\}$ to characterize the task of task node n, where I_n is the input data size, η_n is the processing density measured by the CPU cycles required to process per bit data, and μ_n is the output-input ratio, i.e., the ratio between the output data size and the input data size [15]. For convenience, we use the task n interchangeably for the task of task node n in the following context. Thus, the total required computation resources of task n is $\eta_n I_n$ [15], and the output data size of task n is $\mu_n I_n$.

B. Communication Model

We next introduce the communication model. The communication delay includes two components: the delay caused by transmitting computation tasks and the delay caused by receiving computation results. For cloud computing, we assume that tasks offloaded from one task node to different VMs must be transmitted (received) to (from) the cloud through the same channel simultaneously. Thus, the communication delay caused by offloading computation tasks to the cloud can be represented as

$$T_{n,c}^{t,r}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = \frac{\sum_{k \in \mathcal{C}_n^{\mathcal{V}}} \alpha_{n,k} I_n}{r_n^t} + \frac{\sum_{k \in \mathcal{C}_n^{\mathcal{V}}} \alpha_{n,k} \mu_n I_n}{r_n^r}, \quad (1)$$

where, r_n^t is the transmitting data rate from node n to the cloud, and r_n^r is the receiving data rate from the cloud to node n. The first item and the second item on the right-hand side represent the total transmitting delay and the total receiving delay, respectively.

Similarly, for the helper nodes offloaded computing, the communication delay between the task node n and the helper node m can be written as

$$T_{n,m}^{t,r}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = \frac{\alpha_{n,m} I_n}{r_{n,m}} + \frac{\alpha_{n,m} \mu_n I_n}{r_{m,n}}, \quad (2)$$

where, $r_{n,m}$ and $r_{m,n}$ are the data rate from task node n to helper node m and the data rate from helper node m to task node n, respectively. Similar to [15], [22], we assume that the data rate can be obtained by measurement and is known.

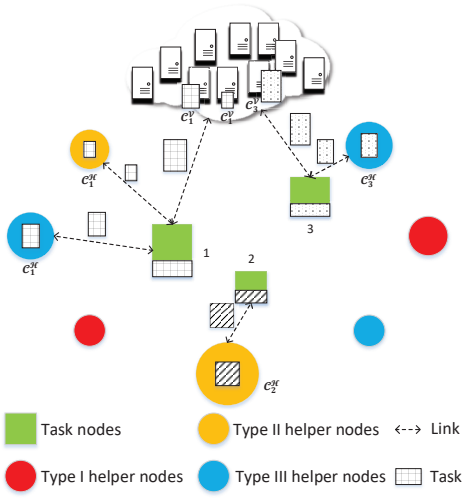

Fig. 1. An illustration of hybrid cloud-fog architecture consisting of multiple FNs and the remote cloud. Here, FNs can be end-user and edge devices, such as smartphones, smart edge routers, or customized FNs. They are with different types (distinguished by colour) and different resource amounts (distinguished by size). They can collaborate with each other to execute computation tasks. Besides, each FN within coverage can access the cloud via a access point (AP) using wired or wireless links.

C. Computation Model

We then introduce the computation model. The computation delay of the local execution can be expressed as

$$T_{n,l}^{c}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = \frac{\alpha_{n,l} \eta_n I_n}{f_n}, \quad (3)$$

where f_n is the CPU frequency (in CPU cycles per second) of task node n.

Accordingly, the computation delay resulted from computation offloading can be written as

$$T_{n,x}^{c}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = \frac{\alpha_{n,x} \eta_n I_n}{f_x}, \ x \in \mathcal{C}_n, \quad (4)$$

where f_x is the CPU frequency of the corresponding helper nodes and VMs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

D. Problem Formulation

We finally formulate the task scheduling problem formally. Table I gives a summary of the key notations. The final latency is composed of three components: the local execution delay, the helper nodes offloaded delay and the cloud offloaded delay, and determined by the maximum of the three.

If the task is executed locally, then the local execution delay can be given by

$$T_{n,l}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T^c_{n,l}(\mathcal{C}_n, \boldsymbol{\alpha}_n), \tag{5}$$

since no communication is necessary for local execution.

If the task is offloaded to helper node m, then the helper node offloaded delay can be represented as the sum of the communication delay and the computation delay:

$$T_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T^{t,r}_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n) + T^c_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n). \tag{6}$$

If the task is offloaded to the cloud, then the cloud offloaded delay is determined by the communication delay and the maximal computation delay of some VM k:

$$T_{n,c}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T^{t,r}_{n,c}(\mathcal{C}_n, \boldsymbol{\alpha}_n) + \max_{k \in \mathcal{C}^{\mathcal{V}}_n} T^c_{n,k}(\mathcal{C}_n, \boldsymbol{\alpha}_n). \tag{7}$$

Therefore, the final latency can be expressed as

$$\begin{aligned} &T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n) \\ &= \max(T_{n,l}(\mathcal{C}_n, \boldsymbol{\alpha}_n), \max_{m \in \mathcal{C}^{\mathcal{H}}_n} T_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n), T_{n,c}(\mathcal{C}_n, \boldsymbol{\alpha}_n)). \end{aligned} \tag{8}$$

Given the system model aforementioned, our objective is to efficiently allocate resources and assign tasks to minimize the total latency of all tasks generated in a timeslot. Similar to many other works [11]–[18], we consider a quasi-static scenario, wherein time is slotted into equivalent intervals. All the new tasks arrived in the same time interval will be processed together at the beginning of next interval. Mathematically, the problem can be formulated as

$$\min_{\{\mathcal{C}_n\}, \{\boldsymbol{\alpha}_n\}} \ \sum_{n \in \mathcal{T}} T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n), \tag{9a}$$

$$s.t.\ C1:\ \mathcal{C}_n \subseteq \mathcal{H} \cup \mathcal{V},\ \forall n \in \mathcal{T}, \tag{9b}$$

$$C2:\ |\mathcal{C}_n| \leq q_n,\ \forall n \in \mathcal{T}, \tag{9c}$$

$$C3:\ \mathcal{C}_n \cap \mathcal{C}_{n'} = \emptyset,\ \forall n, n' \in \mathcal{T}, \tag{9d}$$

$$C4:\ \boldsymbol{\alpha}_n \in \mathbb{R}_+^{|\mathcal{C}_n|},\ \forall n \in \mathcal{T}, \tag{9e}$$

$$C5:\ \mathbf{1}^T \boldsymbol{\alpha}_n = 1,\ \forall n \in \mathcal{T}. \tag{9f}$$

Constraint (9c) ensures that the task node n can offload computation tasks to at most q_n helper nodes and VMs, totally. Constraint (9d) implies that each helper node and VM can only host the computation task from a single task node at every scheduling interval.

The problem (9) is indeed a combinatorial optimization problem, which is NP-hard [23]. In fact, with the explosion of the network size, the optimal solution will become infeasible because of the scalability issue and the lack of global information. Besides, since smart devices want to maximize their own benefits, the optimal solution usually leads to an unstable outcome. Taking all into consideration, we adopt the stable matching theory to develop the DATS algorithm, a suboptimal but stable and scalable algorithm, for this problem.

TABLE I
LIST OF KEY NOTATIONS.

Symbol	Definition
$\mathcal{T}$	The set of task nodes
$\mathcal{H}$	The set of helper nodes
$\mathcal{V}$	The set of VMs
$\mathcal{C}_n$	The coalition of helper nodes and VMs for task node n
q_n	The maximum of $\lvert\mathcal{C}_n\rvert$, quota of task node n
$\mathcal{C}^{\mathcal{H}}_n$	The coalition of helper nodes for task node n
$\mathcal{C}^{\mathcal{V}}_n$	The coalition of VMs for task node n
$\boldsymbol{\alpha}_n$	The task allocation vector of task n
I_n	The input data size of task n
η_n	The processing density of task n
μ_n	The output-input ration of task node n
r^t_n	The transmitting data rate from node n to the cloud
r^r_n	The receiving data rate from the cloud to node n
$r_{n,m}$	The data rate from node n to node m
f_n, f_k	The CPU frequency of node n, VM k
$T^{t,r}_{n,c}$	The communication delay between node n and cloud
$T^{t,r}_{n,m}$	The communication delay between node n and node m
$T^c_{n,l}$	The computation delay of executing tasks on local device
$T^c_{n,x}$	The computation delay of executing tasks on helper node or VM
$T_{n,l}$	The local execution delay
$T_{n,m}$	The helper nodes offloaded delay of offloading task n to helper node m
$T_{n,c}$	The cloud offloaded delay
λ	The processing efficiency, PE

III. Dispersive Stable Task Scheduling

In this section, we first introduce the stable matching theory briefly and transform our problem into a many-to-one matching. We then describe the DATS algorithm in detail, from theories to algorithms. We finally analyze the stability and complexity of the DATS algorithm.

A. Preliminary

We first have a brief overview of the stable matching theory. Matching theory is an important mathematical framework analyzing the formation of mutually beneficial relationships. It has roots in economics but is being fruitfully applied in many other fields, especially the resource allocation problems in networks [24]–[29], for following reasons. First, matching theory can capture the various network features and interpret the complex system requirements, i.e., generalization. Second, matching theory provides low-complexity, self-organized and near-optimal algorithm while guaranteeing the system stability [30]. In this paper, we mainly focus on the two-sided matching, especially the many-to-one matching, since there are two types of nodes, i.e., the task nodes and helpers (helper nodes and VMs), in our problem. In a two-sided matching, there are two distinct sets of agents. A matching is an assignment of agents in one set to agents in the other set. In what follows, we formally define some key concepts in the stable matching theory.

Definition 1. Given the sets $\mathcal{M}$ and $\mathcal{N}$, a *many-to-one matching* is a mapping $\mu : \mathcal{M} \cup \mathcal{N} \to 2^{\mathcal{M} \cup \mathcal{N}}$, such that:

- For each agent $i \in \mathcal{M}$, $|\mu(i)| \leq q_i$, $\mu(i) \subseteq \mathcal{N}$.
- For each agent $j \in \mathcal{N}$, $|\mu(j)| \leq 1$, $\mu(j) \subseteq \mathcal{M}$.
- For each agent i, j, $i \in \mu(j)$ if and only if $j \in \mu(i)$.

where, q_i is the *quota* of the agent i [31].

Stable matching theory is based on the *preference relation*, i.e., the preference ordering of agents in one set over agents

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

in the other set. The preference relation is a complete, anti-symmetric and transitive binary relation [31]. We use $\succ_i$ to denote the preference relation of agent i. If agent i prefers agent j to agent $j^{'}$, then we can use $j \succ_i j^{'}$ to represent this preference ordering. Given a preference relation, the set of partners preferred to the empty set are called *acceptable*. Here, agents in different sets are partners to each other. In the following context, a preference relation only includes the acceptable partners. If no agent is indifferent between any two partners, then we call that the preference relation is *strict*. We refer to a list of preference relations, say $\{\succ_i\}_{i\in\mathcal{M}}$, as a *preference profile*.

Given a set of partners $\mathcal{S}$, let $Ch_a(\mathcal{S})$ denote agent a's *most-preferred* subset of $\mathcal{S}$ under the preference relation of agent a.

Definition 2. A many-to-one matching μ is *individual rational* if and only if $\mu(i) = Ch_i(\mu(i))$ and $\mu(j) \succ_j \emptyset$ $\forall i \in \mathcal{M}$, $\forall j \in \mathcal{N}$ [31].

Individual rationality implies that matches are voluntary [32]. If agent a prefers a proper subset $\mathcal{S}_{\mu(a)} \subsetneq \mu(a)$ over $\mu(a)$, then a will be not willing to be matched with the partners in $\mu(a)\backslash\mathcal{S}_{\mu(a)}$. Note that $\mu(j) \succ_j \emptyset$ is a special case of $\mu(j) = Ch_j(\mu(j))$ when $\mu(j)$ is a singleton.

Definition 3. Given a many-to-one matching μ, a pair of agents (i, j) is called a *pairwise block* of μ if $i \notin \mu(j)$, $j \notin \mu(i)$ such that $j \in Ch_i(\mu(i) \cup j)$ and $i \succ_j \mu(j)$ [31].

Pairwise block indicates that agents in the pairwise block have incentives to deviate from the current matching to form a new matching. Note that $i \succ_j \mu(j)$ is a special case of $i \in Ch_j(\mu(j) \cup i)$ when $\mu(j)$ is a singleton.

Definition 4. A matching μ is *pairwise stable* if it is individual rational and there does not exist a pairwise block in it [31].

If a many-to-one matching μ is pairwise stable, then no participants have incentives to deviate from the matching outcome.

In our task scheduling problem (9), since the task nodes can offload computation tasks to at most q_n helper nodes and VMs while each helper node or VM can only host the task from a single task node, we can naturally transform the problem into a many-to-one matching problem. To be specific, the task nodes and the union of helper nodes and VMs correspond to the two distinct sets of agents aforementioned, i.e. $\mathcal{M}, \mathcal{N}$, respectively. Due to the space limit, we do not explicitly define such a many-to-one matching mathematically. In the following, we will explain the DATS algorithm for problem (9), which is based on the stable matching theory, in detail.

B. DATS Algorithm

1) Overview: We first have an overview of the DATS algorithm. As shown in Fig. 2, the DATS algorithm mainly consists of two phases: the computing resources competition, i.e., PCRC algorithm, and the tasks assignment, i.e., the STS algorithm. And the latter depends on the former. In the beginning, task nodes, helper nodes and VMs discover the resources and collect the required information for calculating PE, which will be discussed in detail next, and constructing the preference profile. This can be accomplished by exchanging hello messages periodically [27]. Then, in the first phase of DATS, i.e., computing resources competition, taking the PEs as input, a global iterative algorithm named PCRC is called to obtain a pairwise stable matching between task nodes and helpers. Particularly, within the PCRC algorithm, the PMCS algorithm is iteratively called to efficiently determine the most-preferred set of helper nodes and VMs for task nodes. In the following tasks assignment phase, given the matching results, the delay-minimization-oriented STS algorithm is called to determine the optimal task allocation vector of each task for minimizing the service delay. The detailed principles and process of DATS algorithm will be discussed in the following.

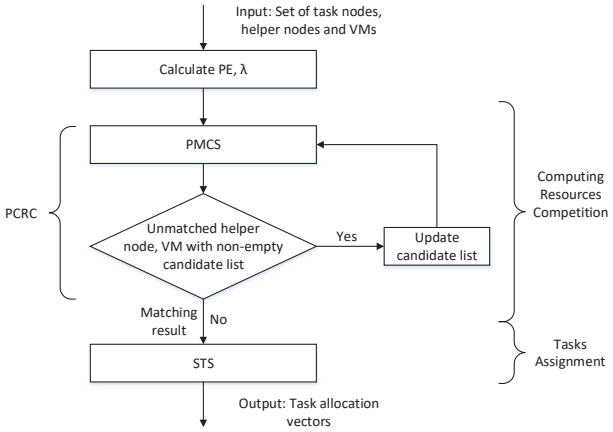

Fig. 2. The flow chart of DATS algorithm.

2) Preference Profile: We next introduce the preference profile. To this end, we define a new concept, PE.

As mentioned above, we can naturally transform the original problem into a many-to-one matching problem. To adopt the stable matching theory to solve our problem, we first need to construct the preference profiles for the task nodes and the union of helper nodes and VMs.

For helper nodes and VMs, their goal is to reduce the delay of task nodes so as to enhance users' experience. Each helper node or VM prefers to help the task node which it can provide the highest processing efficiency, i.e., PE. Since the computation offloading includes two phase: the communication phase and the computation phase, the PE must consider both the communication efficiency and the computation efficiency.

As a result, we define the PE of helper node m when executing task n as

$$\lambda(n, m) = \frac{1}{r_{n,m}} + \frac{\eta_n}{f_m} + \frac{\mu_n}{r_{m,n}}. \tag{10}$$

And the PE of VM k when executing task n solely can be written as

$$\lambda(n, k) = \frac{1}{r_n^t} + \frac{\eta_n}{f_k} + \frac{\mu_n}{r_n^r}. \tag{11}$$

The PE λ signifies the time cost processing per bit data, including transmitting inputs, computing tasks and receiving results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

Therefore, for helper nodes and VMs, we can construct a preference relation over all task nodes as follows.

$$\begin{aligned} n \succ_m n^{'}, \text{ if } \lambda(n,m) < \lambda(n^{'},m),\ \forall m \in \mathcal{H}, \\ n \succ_k n^{'}, \text{ if } \lambda(n,k) < \lambda(n^{'},k),\ \forall k \in \mathcal{V}. \end{aligned} \tag{12}$$

For task nodes, we define the preference relation over all subsets of helper nodes and VMs, instead of single helper node or VM. A task node prefers the coalition of helper nodes and VMs which minimizes the delay. Thus, we can build a preference relation over all coalitions of helper nodes and VMs as follows.

$$\mathcal{C}_n \succ_n \mathcal{C}_n^{'},\ \text{if } T_n(\mathcal{C}_n) < T_n(\mathcal{C}_n^{'}),\ \forall n \in \mathcal{T}, \tag{13}$$

where $T_n(\mathcal{C}_n)$ is the minimum delay that can be achieved by offloading computation tasks to the coalition $\mathcal{C}_n$. That is, $T_n(\mathcal{C}_n)$ satisfies

$$T_n(\mathcal{C}_n) = T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n^*) = \min_{\boldsymbol{\alpha}_n} T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n), \tag{14}$$

where $\boldsymbol{\alpha}_n^*$ is the optimal task allocation vector given the coalition $\mathcal{C}_n$. We define $T_n(\mathcal{C}_n)$ as the minimum delay of task n with coalition $\mathcal{C}_n$.

However, formulas (13) and (14) do not provide us with any insights to efficiently determine the most-preferred coalition of helper nodes and VMs. In the worst case, we have to solve the following optimization problem in (15) for every subset of the helper nodes and VMs set, which is undesirable.

$$\min_{\boldsymbol{\alpha}_n}\ T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n), \tag{15a}$$

$$s.t.\ C1:\ \boldsymbol{\alpha}_n \in \mathbb{R}_+^{|\mathcal{C}_n|}, \tag{15b}$$

$$C2:\ \mathbf{1}^T \boldsymbol{\alpha}_n = 1. \tag{15c}$$

To efficiently determine the most-preferred coalition of helper nodes and VMs for task nodes, we will develop an effective algorithm called PMCS in the following.

3) PMCS Algorithm: We then show the details of PMCS algorithm in this section. To proceed, we first propose four propositions. Proposition 1 and proposition 2 reveal the structure of the optimal task allocation vector given the coalition $\mathcal{C}_n$, while proposition 3 and proposition 4 show how to find the most-preferred coalition of helper nodes and VMs.

Proposition 1. For a given $\mathcal{C}_n$, $\boldsymbol{\alpha}_n$ is a feasible task allocation vector. There exists a feasible task allocation vector $\boldsymbol{\alpha}_n^{'}$, satisfying $T_{n,k'}^c(\mathcal{C}_n, \boldsymbol{\alpha}_n^{'}) = T_{n,k}^c(\mathcal{C}_n, \boldsymbol{\alpha}_n^{'})$ for $\forall k,\ k^{'} \in \mathcal{C}_n^{\mathcal{V}}$, being at least as good as $\boldsymbol{\alpha}_n$, i.e. $T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n^{'}) \le T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n)$.

Proposition 1 implies that the (optimal) task allocation vector should be VMs-load-balanced, i.e., $T_{n,k'}^c(\mathcal{C}_n, \boldsymbol{\alpha}_n^{'}) = T_{n,k}^c(\mathcal{C}_n, \boldsymbol{\alpha}_n^{'})$ for $\forall k,\ k^{'} \in \mathcal{C}_n^{\mathcal{V}}$. We refer to such a $\boldsymbol{\alpha}_n^{'}$ as a VMs-load-balanced version of $\boldsymbol{\alpha}_n$. For convenience, we only consider the VMs-load-balanced version of task allocation vectors in the following context.

Proposition 2. For a given $\mathcal{C}_n$, $\boldsymbol{\alpha}_n$ is the optimal task allocation vector, if it satisfies that the local execution delay, the helper nodes offloaded delay, and the cloud offloaded delay are identical, i.e. $T_n(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T_{n,l}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T_{n,c}(\mathcal{C}_n, \boldsymbol{\alpha}_n)$, $\forall m \in \mathcal{C}_n^{\mathcal{H}}$.

Algorithm 1 Progressive Most-preferred Coalition Selection (PMCS) Algorithm

Require: $\mathcal{A}_n$, q_n, $\lambda(n,x)$, $\forall x \in \mathcal{A}_n$.
Ensure: $\mathcal{C}_n$.

```
if |A_n| ≤ q_n then
    C_n = A_n.
else
    Maintain two sequences A_n^H and A_n^V for the helper nodes and VMs, respectively.
    Sort A_n^H and A_n^V in ascending order of PE λ.
    Initialize A_n^H and A_n^V as the first min(|A_n^H|, q_n) and min(|A_n^V|, q_n) elements, respectively.
    Maintain two pointers p_1, p_2 pointing to the end of A_n^H and A_n^V, respectively. m_{p_1} and k_{p2} are the corresponding helper node and VM.
    while |A_n^H| + |A_n^V| > q_n do
        if |A_n^V| = 1 then
            if λ(n, m_{p_1}) < λ(n, k_{p_2}) then
                Remove k_{p_2} from A_n^V.
            else
                Remove m_{p_1} from A_n^H.
            end if
        else
            Calculate the PE λ(n, A_n^V) and λ(n, A_n^{V'}), where A_n^{V'} = A_n^V \ k_{p_2}.
            if 1 + λ(n,m_{p_1})/λ(n,A_n^{V'}) − λ(n,m_{p_1})/λ(n,A_n^V) > 0 then
                Remove k_{p_2} from A_n^V.
                Point p_2 to the end of A_n^V.
            else
                Remove m_{p_1} from A_n^H.
                Point p_1 to the end of A_n^H.
            end if
        end if
    end while
    C_n = A_n^H ∪ A_n^V.
end if
Return C_n.
```

Proposition 2 indicates that the optimal task allocation vector should be task node, helper nodes and cloud load-balanced, i.e., $T_{n,l}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T_{n,m}(\mathcal{C}_n, \boldsymbol{\alpha}_n) = T_{n,c}(\mathcal{C}_n, \boldsymbol{\alpha}_n)$, $\forall m \in \mathcal{C}_n^{\mathcal{H}}$.

Proposition 3. For an arbitrary coalition of helper nodes and VMs, say $\mathcal{C}_n$, with $|\mathcal{C}_n| < q_n$, we can reduce the final minimum delay $T_n(\mathcal{C}_n)$ by adding an arbitrarily available helper node or VM to the coalition $\mathcal{C}_n$.

Proposition 3 shows that more helper nodes or VMs (no more than the quota) participate in the task execution, lower delay can be achieved.

Proposition 4. For an arbitrary coalition of helper nodes and VMs, say $\mathcal{C}_n$, with $|\mathcal{C}_n| = q_n$, we can reduce the final minimum delay $T_n(\mathcal{C}_n)$ by

Case 1: replacing an arbitrary helper node $m_r \in \mathcal{C}_n^{\mathcal{H}}$ with another available helper node m_s satisfying $\lambda(n, m_s) < \lambda(n, m_r)$.

Case 2: replacing an arbitrary VM $k_r \in \mathcal{C}_n^{\mathcal{V}}$ with another

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

available VM k_s satisfying $\lambda(n,k_s) < \lambda(n,k_r)$. Actually, it is equivalent to $f^c_{k_s} > f^c_{k_r}$ in this case.

Case 3: replacing an arbitrary helper node $m_r \in \mathcal{C}_n^{\mathcal{H}}$ with another available VM k_s satisfying $\frac{\lambda(n,m_r)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}'})} - \frac{\lambda(n,m_r)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}})} - 1 > 0$. $\mathcal{C}_n^{\mathcal{V}'} = \mathcal{C}_n^{\mathcal{V}} \cup k_s$. When $\mathcal{C}_n^{\mathcal{V}} = \emptyset$, then it degenerates to $\lambda(n,k_s) < \lambda(n,m_r)$.

Case 4: replacing an arbitrary VM $k_r \in \mathcal{C}_n^{\mathcal{V}}$ with another available helper node m_s satisfying $1+\frac{\lambda(n,m_s)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}'})} - \frac{\lambda(n,m_s)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}})} > 0$. $\mathcal{C}_n^{\mathcal{V}'} = \mathcal{C}_n^{\mathcal{V}} \backslash k_r$. When $\mathcal{C}_n^{\mathcal{V}'} = \emptyset$, then it degenerates to $\lambda(n,m_s) < \lambda(n,k_r)$.

$\lambda(n,\mathcal{C}_n^{\mathcal{V}})$ is the PE of the coalition $\mathcal{C}_n^{\mathcal{V}}$ when executing task n and defined as

$$\lambda(n,\mathcal{C}_n^{\mathcal{V}}) = \frac{1}{r_n^t} + \frac{\eta_n}{\sum\limits_{k\in\mathcal{C}_n^{\mathcal{V}}} f_k} + \frac{\mu_n}{r_n^r}. \tag{16}$$

Proof: See appendix A-D. ∎

Based on the aforementioned four propositions, we can develop the PMCS algorithm to efficiently determine the task nodes' most-preferred coalition of helper nodes and VMs in polynomial time. To be specific, if the helper nodes and VMs are less than the quota, then the most-preferred coalition is just the set of helper nodes and VMs according to the proposition 3 (line 1-2). Otherwise, we maintain two sequences: the helper nodes sequence and the VMs sequence (line 4). We first sort the helper nodes and VMs in ascending order of the PE λ (line 5). Then, we take the first quota elements of the helper nodes sequence and the VMs sequence, respectively. If the helper nodes or VMs are less than the quota, then take all of them (line 6). Next, we maintain two pointers pointing to the last element of the helper nodes sequence and the VMs sequence, respectively (line 7). If these two elements satisfy the case 3 in the proposition 4, then we remove the last element from the helper nodes sequence (line 13, line 21-22). Otherwise, they must be in the case 4, and we remove the last element from the VMs sequence (line 11, line 18-19). The algorithm iterates until the number of helper nodes and VMs equals to the quota. The whole procedure runs as shown in algorithm 1.

4) PCRC Algorithm: After defining the PE concept and developing the PMCS algorithm to construct the preference profile for task nodes, helper nodes and VMs, we formally introduce the PCRC algorithm to allocate computing resources, i.e. helper nodes and VMs, to task nodes. The whole algorithm is based on the deferred acceptance (DA) algorithm [33] and runs as follows. In the first round, each helper node or VM applies to the favourite task node in its candidate list (line 6-9), where candidate lists represent the available and acceptable partners. Among all applicants, each task node runs the PMCS algorithm to determine the most-preferred coalition of helper nodes and VMs (line 12). Each task node puts the most-preferred coalition in the waiting list and rejects the others (line 13-14). In the following rounds, each rejected helper node or VM applies to its most-preferred task node which has never rejected it before (line 6-9). Each task node with new applicants (nonempty current proposer list) updates its most-preferred coalition of helper nodes and VMs and rejects the others (line 12-14). This process is repeated until all helper nodes and VMs have exhausted their applications. The whole procedure runs as shown in algorithm 2. For detailed principle of the above algorithm, please refer to [33]. Notably, the whole algorithm can be implemented in a distributed way [28], [33].

Algorithm 2 Progressive Computing Resources Competition (PCRC) Algorithm

Require: $\lambda(n,x)$, $\forall n \in \mathcal{T},\ x \in \mathcal{H} \cup \mathcal{V}$.
Ensure: μ.
1: Initialization:
2: Construct the candidate lists (preference profile) $\mathcal{L}_m$, $\mathcal{L}_k$, $\forall m \in \mathcal{H}$, $k \in \mathcal{V}$, for helper nodes and VMs.
3: $\mu(n) = \emptyset$, $\mu(m) = \emptyset$, $\mu(k) = \emptyset$, $\forall n \in \mathcal{T}$, $\forall m \in \mathcal{H}$, $k \in \mathcal{V}$. The waiting list $\mathcal{W}_n = \emptyset$, the current proposer list $\mathcal{P}_n = \emptyset$, $\forall n \in \mathcal{T}$.
4: **while** $\exists$ unmatched helper node, VM with non-empty candidate list **do**
5: **for all** unmatched helper node m, VM k with non-empty candidate list **do**
6: n := the most-preferred task node in the candidate list.
7: Helper node m or VM k proposes to task node n.
8: Add helper node m or VM k to task node n's current proposer list $\mathcal{P}_n$.
9: Remove task node n from the helper node m's or VM k's candidate list.
10: **end for**
11: **for all** task node n with nonempty current proposer list $\mathcal{P}_n$ **do**
12: Call the **PMCS** algorithm to select the most-preferred coalition $\mathcal{C}_n$ from the set $\mathcal{W}_n \cup \mathcal{P}_n$ with $\mathcal{A}_n = \mathcal{W}_n \cup \mathcal{P}_n$.
13: Update waiting list, i.e. $\mathcal{W}_n = \mathcal{C}_n$.
14: Reject other applicants and clear the current proposer list.
15: **end for**
16: **end while**
17: **for all** task node $n \in \mathcal{T}$ **do**
18: $\mu(n) = \mathcal{W}_n$.
19: **end for**
20: **for all** helper node $m \in \mathcal{H}$, VM $k \in \mathcal{V}$ **do**
21: $\mu(m) = \{n | m \in \mathcal{W}_n, \forall n \in \mathcal{T}\}$.
22: $\mu(k) = \{n | k \in \mathcal{W}_n, \forall n \in \mathcal{T}\}$.
23: **end for**
24: Return μ.

5) STS Algorithm: We finally describe the STS algorithm for task assignment. Once obtaining the matching result between task nodes and helpers, we can derive the STS algorithm to determine the optimal task allocation vector of each task node according to the proposition 1 and 2 above. Due to the space constraint, we omit the details of derivation here.

$$\alpha_{n,l} = \frac{1}{1 + \sum\limits_{m\in\mathcal{C}_n^{\mathcal{H}}} \frac{\lambda(n,l)}{\lambda(n,m)} + \frac{\lambda(n,l)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}}, \tag{17}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

Algorithm 3 Synchronized Task Scheduling (STS) Algorithm

Require: μ, $\lambda(n,x)$, $\forall n \in \mathcal{T},\ x \in \mathcal{H} \cup \mathcal{V}$.
Ensure: $\{\boldsymbol{\alpha}_n\}_{n\in\mathcal{T}}$.
1: **for all** task node $n \in \mathcal{T}$ **do**
2: Calculate $\alpha_{n,l}$ according to (17).
3: Calculate $\alpha_{n,m},\ \forall m \in \mathcal{C}_n^{\mathcal{H}}$ according to (18).
4: Calculate $\alpha_{n,k},\ \forall k \in \mathcal{C}_n^{\mathcal{V}}$ according to (19), (20).
5: **end for**
6: Return $\boldsymbol{\alpha}_n, \forall n \in \mathcal{T}$.

$$\alpha_{n,m} = \frac{1}{1 + \frac{\lambda(n,m)}{\lambda(n,l)} + \sum\limits_{m' \in \mathcal{C}_n^{\mathcal{H}} \setminus m} \frac{\lambda(n,m)}{\lambda(n,m')} + \frac{\lambda(n,m)}{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}},\ \forall m \in \mathcal{C}_n^{\mathcal{H}}, \tag{18}$$

$$\sum_{k\in\mathcal{C}_n^{\mathcal{V}}} \alpha_{n,k} = \frac{1}{1 + \frac{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}{\lambda(n,l)} + \sum\limits_{m\in\mathcal{C}_n^{\mathcal{H}}} \frac{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}{\lambda(n,m)}}, \tag{19}$$

and,

$$\alpha_{n,k} = \frac{1}{1 + \frac{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}{\lambda(n,l)} + \sum\limits_{m\in\mathcal{C}_n^{\mathcal{H}}} \frac{\lambda(n,\mathcal{C}_n^{\mathcal{V}})}{\lambda(n,m)}} \times \frac{f_k}{\sum\limits_{k\in\mathcal{C}_n^{\mathcal{V}}} f_k},\ \forall k \in \mathcal{C}_n^{\mathcal{V}} f_k, \tag{20}$$

where $\lambda(n,l)$ is the local PE of task node n and defined as

$$\lambda(n,l) = \frac{\eta_n}{f_n}. \tag{21}$$

C. Stability Analysis

Proposition 5. The DATS algorithm is pairwise stable.

Proof: We complete the proof by contradiction. Suppose there exists a block pair of helper node m and task node n, i.e., $n \notin \mu(m)$, $m \notin \mu(n)$, $m \in Ch_n(\mu(n) \cup m)$ and $n \succ_m \mu(m)$. Since $n \succ_m \mu(m)$ and $m \notin \mu(n)$, helper node m must propose to task node n which rejects m afterwards. And according to the principle of the PMCS algorithm, task node n would never accept helper node m even if helper node m proposed to task node n again. Thus, $m \notin Ch_n(\mu(n) \cup m)$. In conclusion, μ is pairwise stable. The proof for the case of VM k and task node n is the same and omitted here. ■

Actually, the DATS algorithm is pairwise stable because the rejections of task nodes are definitive. Besides, the proposed algorithm with helper nodes and VMs proposing produces an optimal stable matching for helper nodes and VMs [32]. Since helper nodes and VMs are voluntary to participate in, it may be convincing to let helper nodes and VMs be the proposing side.

D. Complexity Analysis

Proposition 6. The DATS algorithm is guaranteed to converge, and the running time is $O(N(M+K)^2 \log(M+K))$.

Proof: The convergence and computational complexity of the DATS algorithm are determined by the computing resources competition phase, i.e., the PCRC algorithm. In each round of the PCRC algorithm, there exists at least one unmatched helper node or VM proposing to one task node and removing it from its candidate list. Thus, each helper node or VM will eventually become matched or exhaust its applications. Therefore, the proposed DATS algorithm is guaranteed to converge.

In the worst case where only one application happens in each round, the PCRC algorithm will iterate for $(M+K)N$ times. In addition, the PMCS algorithm mainly contains the sort operation and the while loop. For the sort operation, its running time is $O((M+K)\log(M+K))$. In each iteration of while loop, we remove an element from the helper nodes sequence or the VMs sequence. Since the number of helper nodes and VMs is reduced from $2q_n$ to q_n after the while loop, the while loop will run for $\max\limits_n q_n$ times. Considering that $\max\limits_n q_n$ is usually much less than $M+K$, the PMCS algorithm costs $O((M+K)\log(M+K))$ time.

In conclusion, the running time of the proposed DATS algorithm is $O(N(M+K)^2 \log(M+K))$, which is polynomial time. ■

TABLE II
SIMULATION PARAMETERS.

Parameter	Value
$M+N$ (# of FNs)	200
K (# of VMs)	50
q_n (quota of computation offloading)	[1, 2, 3, 4, 5, 6] [12]
f_n, f_m (computational capability of FNs)	[1.0, 1.5, 1.8, 2.0, 2.5] GHz
f_k (computational capability of VMs)	[10, 15, 20, 25, 30, 35] GHz
B (bandwidth of subcarrier)	[15, 30, 45, 60, 75, 90] KHz
p_t (transmitting power)	100-350mw, for FNs 20w, for base station
N_0 (noise power spectral density)	-174 dBm/Hz
I_n (input data size)	$[1000, 6000]$KB
processing density of light tasks	500 cycle/bit
processing density of medium tasks	2000 cycle/bit
processing density of heavy tasks	3000 cycle/bit
μ_n (output-input-ratio)	[0.01, 0.5]

IV. PERFORMANCE EVALUATION

In this section, simulation results of the proposed algorithm are presented and analyzed in comparison with several baseline schemes. For simulations, we consider a scenario where 200 FNs are uniformly scattered within a 200m×200m square area. Each FN can connect to neighboring FNs and the remote cloud via a BS in the middle of the area. Given the small coverage, an OFDMA system is considered here so that interference among FNs can be ignored. The computational capability of a FN is randomly selected from the set $[1.0, 1.5, 1.8, 2.0, 2.5]$ GHz to account for the heterogeneous computing capability of FNs. There are $K = 50$ VMs available at the cloud [34]. Each FN can offload tasks to multiple neighboring FNs and VMs which are no more than the quota.

Each FN can use a subcarrier to transmit tasks and receive results. The data rate can be expressed as

$$r = B\log_2(1 + \frac{gp_t}{BN_0}) \tag{22}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

where, B is the bandwidth of the channel. g is the channel gain. p_t is the transmitting power and N_0 is the noise power spectral density. The channel is given by a large scale fading model [35]:

$$PL = 20\log(d^{km}) + 20\log(B^{kHz}) + 32.45(dB). \quad (23)$$

To take the heterogeneity of tasks into consideration, three different types of tasks are considered here: the light workload task (e.g., video transcoding), the medium workload task(e.g., chess game) and the heavy workload task (e.g., face recognition), which are in accordance with the real measurements in practice [34]. All parameters are summarized in TABLE II.

The locations of FNs vary in each task scheduling interval to mimic dynamics of nodes. In each interval, task nodes are randomly chosen from whole FNs according to the active ratio, i.e., the percentage of task nodes, which is an indicator of network workload. Tasks are generated randomly based on parameters aforementioned. Here, the initialization phase, which can be accomplished during the network access, is considered. We set the delay caused by the initialization phases as 200 ms [36], [37]. All numerical results are averaged over 1000 simulations.

Fig. 3 compares the performance of the random task scheduling (RTS) scheme, the indexed task scheduling (ITS) scheme and the proposed DATS scheme. For the RTS scheme, helper nodes and VMs are first randomly allocated to task nodes, and then task nodes run the STS algorithm to determine the optimal task allocation vectors. For the ITS scheme, tasks nodes first run the PMCS algorithm to choose the most-preferred coalition in ascending order of the node index, and then run the STS algorithm to determine the optimal task allocation vectors. Besides, the same extra time cost of the initialization phase is considered for both the RTS scheme and the ITS scheme. As shown in Fig. 3, the DATS scheme can achieve higher latency reduction ratio compared with the RTS scheme and the ITS scheme. Here, the latency reduction ratio is defined as the total latency reduction compared with the total latency of local execution. To be specific, the DATS scheme can reduce about 65% latency when the active ratio is low, i.e., less than 0.3. When the active ratio is medium, i.e. 0.4-0.5, the DATS scheme can reduce more than 15% latency over the RTS scheme and the ITS scheme. Even though when the active ratio is high, the DATS scheme can still achieve additional $5\% - 10\%$ latency reduction compared with the RTS scheme and the ITS scheme. The reason is: (i) the PE incorporates both the computing resources and communication capabilities of helper nodes and VMs, and (ii) the DATS scheme enables task nodes to select the optimal coalitions from helper nodes and VMs and guarantees a stable outcome. As a result, the DATS scheme can achieve effective tradeoff between computing resources and communication capabilities of helper nodes and VMs among different task nodes.

It is worth noting that when the active ratio is low, the computing resources are abundant. Thus, the RTS scheme, the ITS scheme and the DATS scheme can all achieve a superior performance. However, since the ITS scheme and the DATS scheme can both select the optimal coalition for task nodes, both of them achieve higher latency reduction ratio than the RTS scheme. This also explains why the ITS scheme can achieve nearly the same performance with the DATS scheme under low active ratio and substantiates the efficiency of the PMCS algorithm. On the other hand, when the network workload is heavy, the computing resources are scarce, and hence all three schemes perform badly. However, when the workload is medium, the tradeoff between computing resources and communication capabilities of helper nodes and VMs among different task nodes becomes the key. Consequently, the superiority of the DATS scheme becomes more pronounced. Besides, it is worth noticing that there is a steep drop in performance of the ITS scheme when the active ratio increases from 0.2 to 0.3. It is because that the ITS scheme leads to the unfairness of the resources allocation among task nodes when the network load becomes heavier. This also indirectly demonstrates the superiority of the two-sided matching and the stable outcome of the DATS algorithm.

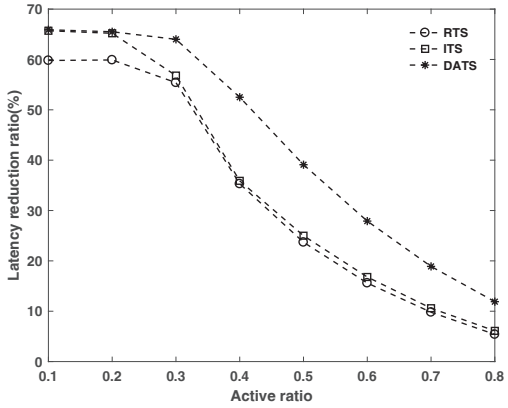

Fig. 3. DATS verse ITS, RTS (q_n: 1-6, B: 15 KHz, p_t: 100-350 mw, f_k: 10 GHz, I_n: 1000-6000 KB).

Due to the superior performance of the DATS, we investigate the relationship between the performance of DATS and different system configurations in the following Fig. 4-Fig. 9.

Fig. 4 illustrates latency versus quota under different active ratios. The cases of quota being zero imply that tasks are computed locally. As shown in Fig. 4, the optimal quota, i.e., quota making latency minimum, varies with the active ratio. Generally speaking, the optimal quota decreases with the increase of active ratio. For instance, the optimal quota is 6 when the active ratio is 0.1, but it reduces to 2 if the active ratio rises up to 0.5. This is because that computing resources are abundant when active ratio is low. In this case, the more computing resources are available, i.e., quota is larger, smaller the latency is. While, computing resources become inadequate when network workload becomes heavy. As a result, the large quota will result in the unfairness of computing resources allocation between task nodes so as to decrease the performance of system. This also explains why the latency increases with quota when workload is heavy. From Fig. 4, we can conclude that a fair allocation of computing resources between task nodes may be a good choice.

Fig. 5 demonstrates the effects of bandwidth on latency under different active ratios. The cases of bandwidth being

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2884720, IEEE Internet of Things Journal

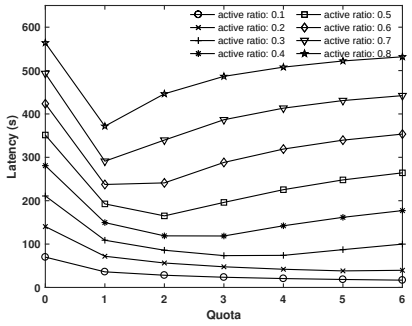

Fig. 4. Latency versus quota under different active ratio (B: 15 KHz, p_t: 100 mw, f_k: 10 GHz, I_n: 1000 KB).

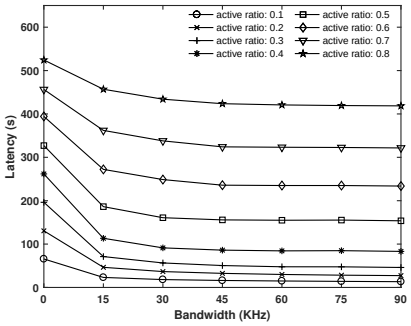

Fig. 5. Latency versus bandwidth under differnet active ratio (q_n: 3, p_t: 100 mw, f_k: 10 GHz, I_n: 1000 KB).

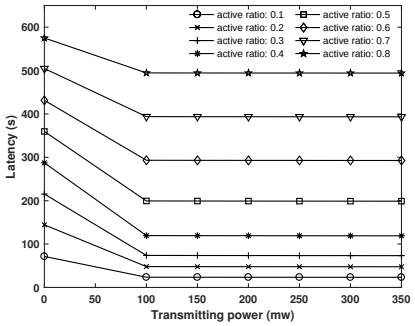

Fig. 6. Latency versus transmitting power under different active ratio (B: 15 KHz, q_n: 3, f_k: 10 GHz, I_n: 1000 KB).

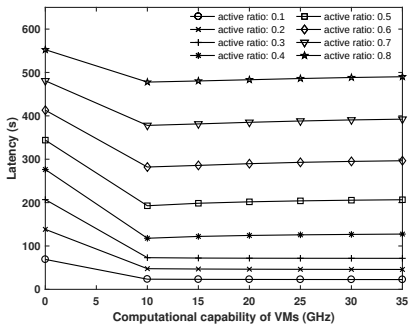

Fig. 7. Latency versus computational capability of VMs under different active ratio (q_n: 3, B: 15 KHz, p_t: 100 mw, I_n: 1000 KB).

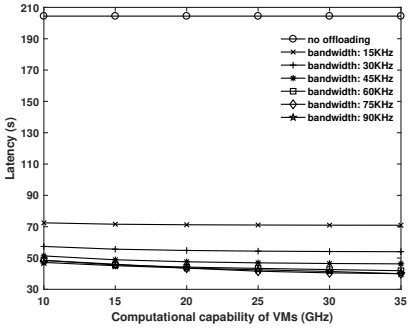

Fig. 8. Latency versus computational capability of VMs under different bandwidth (q_n: 3, p_t: 100 mw, active ratio: 0.3, I_n: 1000 KB).

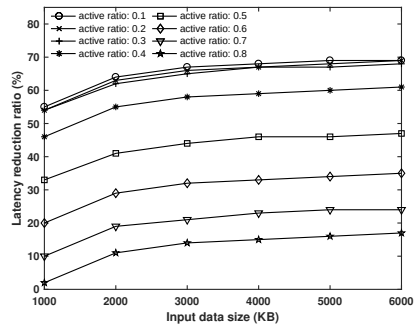

Fig. 9. Latency reduction ration versus input data size under different active ratio (B: 15 KHz q_n: 3, p_t: 100 mw).

zero imply that tasks are computed locally. As illustrated in Fig. 5, the latency decreases with the increase of bandwidth and levels off under all active ratios. The reason is that the communication capability, i.e., data rate, increases with the bandwidth at the beginning. However, when the communication capability increases to certain extent, the efficiency of computation offloading is limited by the computation capability of helper nodes and cloud.

Fig. 6 depicts the relationship between latency and transmitting power of FNs under different active ratios. The cases of transmitting power being zero imply that tasks are computed locally. From Fig. 6, we can observe that the transmitting power of FNs with rather large values has little effect on latency. This is because in this region, the computation capability dominates the latency instead of the transmission power.

Fig. 7 shows latency under different computational capability of VMs and active ratios. The cases of computational capability of VMs being zero imply no offloading. It is interesting to note that the computational capability of VMs has little effect on latency, which is counter-intuitive. It may be attributable to the fact that the communication capability is the bottleneck of latency. Even though the computational capability of cloud is high, the final efficiency of computation offloading is decided by the communication capability in this case. What's more, it is observed that the latency increases slightly with the computational capability of VMs under high active ratios. This may be caused by that the increase of VMs' computational capability will exacerbate the unfairness of computing resources allocation between task nodes in our algorithm, which will decrease the efficiency of computation offloading as mentioned above. Such an unfair resource allocation stands out when computing resources are scare, i.e., active ratio is high.

Fig. 8 substantiates the inter-constraint relationship between communication capability and computation capability. As demonstrated in Fig. 8, the computational capability of VMs nearly has no effect on latency when bandwidth is under 30 KHz. In contrast, the bandwidth has an significant effect on latency. The reason is that, in this case, the computation capability is abundant so that the efficiency of computation offloading is decided by the communication capability. Similarly, when bandwidth is over 30 KHz, the bandwidth nearly has no effect on latency because the communication capability is abundant. As a result, the computational capability of VMs dominates the efficiency of computation offloading. It can be observed that the decline degree of latency with the VMs' computational capability under 90 KHz bandwidth becomes larger, compared with that under 15 KHz bandwidth.

Fig. 9 compares the effect of input data size on the efficiency of computation offloading under different active ratios. From Fig. 9, we can see that the latency reduction ratio decreases with the decrease of input data size. It is due to the fact that the local execution time decreases with the input data size. As a result, the influence brought about by the extra time cost of the

initialization phase becomes significant. Besides, we observe that the latency reduction ratio decreases with the increase of active ratio. This is because that less FNs are available to participate in computation offloading as the active ratio increases. It is suggested that local execution may be a better choice under light tasks, i.e., small data size.

V. Conclusion

In this paper, we investigated a service delay-minimization task scheduling problem in a typical heterogeneous fog network, where dispersive task nodes could simultaneously offload tasks to multiple neighboring helper nodes and the cloud so that each task could be executed in parallel. We defined a new concept named PE to incorporate computing resources and communication capabilities, and further proposed a scalable, stable and decentralized algorithm called DATS which consists of two key components: (i) the PCRC algorithm for computing resources allocation and (ii) the STS algorithm for tasks assignment. Concretely, we first formulated the computing resources allocation as a many-to-one matching and developed the PMCS algorithm based on the PE to tackle the critical difficulties in constructing the preference profile. Then, taking PMCS as a key step, we proposed the PCRC to obtain a pairwise stable resources allocation result based on the DA algorithm. Finally, we derived the STS to determine the optimal task assignment for each task, based on the PE and computing resources allocation result. Extensive simulations were conducted to demonstrate the performance of the proposed DATS algorithm. Theoretical proofs and simulation results showed that the DATS algorithm achieved effective tradeoff between computing resources and communication capabilities, thus significantly reducing service delay in heterogeneous fog networks.

In our future work, we will investigate how to take the energy consumption into consideration when constructing the preference profile to deal with the energy-latency tradeoff problem in heterogeneous fog networks.

References

[1] Y. Yang, J. Xu, G. Shi, and C.-X. Wang, *5G wireless systems: simulation and evaluation techniques*. Springer, 2017.

[2] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, no. 5, pp. 2795–2808, 2016.

[3] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, ACM, 2012.

[5] M. Chiang and T. Zhang, "Fog and iot: an overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[6] J. Ni, K. Zhang, X. Lin, and X. S. Shen, "Securing fog computing for internet of things applications: challenges and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 601–628, 2017.

[7] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2017.

[8] V. B. C. d. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *Communications (ICC), 2016 IEEE International Conference on*, pp. 1–5, IEEE, 2016.

[9] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: towards minimizing delay in the internet of things," in *Edge Computing (EDGE), 2017 IEEE International Conference on*, pp. 17–24, IEEE, 2017.

[10] W. Masri, I. Al Ridhawi, N. Mostafa, and P. Pourghomi, "Minimizing delay in iot systems through collaborative fog-to-fog (f2f) communication," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*, pp. 1005–1010, IEEE, 2017.

[11] Y.-Y. Shih, W.-H. Chung, A.-C. Pang, T.-C. Chiu, and H.-Y. Wei, "Enabling low-latency applications in fog-radio access networks," *IEEE network*, vol. 31, no. 1, pp. 52–58, 2017.

[12] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[13] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 615–624, IEEE, 2017.

[14] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: an energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3887–3901, 2016.

[15] X. Chen and J. Zhang, "When d2d meets cloud: hybrid mobile task offloadings in fog computing," in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[16] N. T. Ti and L. B. Le, "Computation offloading leveraging computing resources from edge cloud and mobile peers," in *Communications (ICC), 2017 IEEE International Conference on*, pp. 1–6, IEEE, 2017.

[17] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," *IEEE Access*, vol. 5, pp. 21355–21367, 2017.

[18] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "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, 2016.

[19] S. Zhao, Y. Yang, Z. Shao, X. Yang, H. Qian, and C.-X. Wang, "Femos: fog-enabled multitier operations scheduling in dynamic wireless networks," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1169–1183, 2018.

[20] Y. Yang, S. Zhao, W. Zhang, Y. Chen, X. Luo, and J. Wang, "Debts: delay energy balanced task scheduling in homogeneous fog networks," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2094–2106, 2018.

[21] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, and M.-T. Zhou, "Meets: maximal energy efficient task scheduling in homogeneous fog networks," *IEEE Internet of Things Journal*, 2018.

[22] H. Shah-Mansouri and V. W. Wong, "Hierarchical fog-cloud computing for iot systems: a computation offloading game," *IEEE Internet of Things Journal*, 2018.

[23] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[24] Y. Gu, W. Saad, M. Bennis, M. Debbah, and Z. Han, "Matching theory for future wireless networks: fundamentals and applications," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 52–59, 2015.

[25] A. Leshem, E. Zehavi, and Y. Yaffe, "Multichannel opportunistic carrier sensing for stable channel access control in cognitive radio systems," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 1, pp. 82–95, 2012.

[26] F. Pantisano, M. Bennis, W. Saad, S. Valentin, and M. Debbah, "Matching with externalities for context-aware user-cell association in small cell networks," in *Globecom Workshops (GC Wkshps), 2013 IEEE*, pp. 4483–4488, IEEE, 2013.

[27] K. Hamidouche, W. Saad, and M. Debbah, "Many-to-many matching games for proactive social-caching in wireless small cell networks," in *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2014 12th International Symposium on*, pp. 569–574, IEEE, 2014.

[28] H. Xu and B. Li, "Seen as stable marriages," in *INFOCOM, 2011 Proceedings IEEE*, pp. 586–590, IEEE, 2011.

[29] Y. Chen, L. Jiang, H. Cai, J. Zhang, and B. Li, "Spectrum matching," in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pp. 590–599, IEEE, 2016.

[30] Z. Han, Y. Gu, and W. Saad, *Matching theory for wireless networks*. Springer, 2017.

[31] F. Echenique and J. Oviedo, "A theory of stability in many-to-many matching markets," *Theoretical Economics*, vol. 1, pp. 233–273, 2006.
[32] U. Kamecke, "Two sided matching: a study in game-theoretic modeling and analysis," 1992.
[33] 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.
[34] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.
[35] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and cpu time allocation for mobile edge computing," in *Global Communications Conference (GLOBECOM), 2016 IEEE*, pp. 1–6, IEEE, 2016.
[36] E. U. T. R. Access, "Requirements for support of radio resource management(release 10) 3gpp ts 36.133," *V10*, vol. 3, 2011.
[37] T. Sakurai and H. L. Vu, "Mac access delay of ieee 802.11 dcf," *IEEE Transactions on Wireless Communications*, vol. 6, no. 5, 2007.