An Energy-Aware High Performance Task Allocation Strategy in Heterogeneous Fog Computing Environments

Keke Gai, *Senior Member, IEEE*, Xiao Qin, *Senior Member, IEEE*, and Liehuang Zhu, *Member, IEEE*

Abstract—Combining the *Internet-of-Things* (IoT) technology with cloud computing is a significant alternative for powering the utilization of computing resources in a connected environment. A grand challenge in communications is raised by the emergence of big data, due to the large-sized data transmissions and frequent data exchanges. Applying fog computing is considered an option for resolving the communication challenge. However, a high extent of available heterogeneous computing attached to fog computing servers leads to a restriction of the resource management. This Article addresses the resource management issue by proposing a novel approach - named *Energy-aware Fog Resource Optimization* (EFRO) model- to optimizing the utilization of connected devices in fog computing. We develop a heuristic algorithm minimizing both energy cost and time consumption in a holistic way. A salient feature of EFRO lies in the integration of the standardization and smart shift operations fueled by a hill-climbing mechanism to produce near-optimal resource allocation solutions. Experimental results demonstrate that our EFRO is adroit at making near-optimal decisions in managing resources in fog computing environments. In particular, EFRO boosts the energy efficiency of the existing *MESF* and *RR* schemes by 54.83 and 71.28 percent, respectively. EFRO shortens DECM's allocation-generation time by up to a factor of 507.

Index Terms—Fog computing, energy-aware, resource management, heterogeneous computing, allocation strategy

1 Introduction

The widespread application of contemporary cloud computing has significantly promoted the close relationship between computing resources and users through networks. The formulation of connected environments further integrates cloud computing with other technical dimensions like the *Internet-of-Things* technology or *IoT* [1], [2]. One of the major benefits of combining cloud computing with IoT is boosting the utilization of distributed devices running on centralized computing platforms. IoT furnishes devices with networking interconnects for data and task outsourcing, whereas cloud computing offers powerful centralized computing capability [3]. The aggressive increase of cloud-based applications and IoT devices has become a main trend reshaping industrial operations and business models in recent years [4]. In a cloud-IoT model, user-side devices are in a strong likelihood to play a role of service interface that receives and responds service operations; remote cloud servers are in charge of handling heavy workload behind the scene [5].

Despite the enhancement deriving from clouds and IoT, a challenge is raised by the rapid growth of big data in a wide range of applications. Frequent data exchanges and transmissions demand a high bandwidth and a stable connection. Network latency is enlarged when the data size of transmission packets becomes large [6], [7]. Such a latency obstacle is pronounced because centralized cloud-based computing requires consistent data exchanges and synchronizations. More often than not, a central cloud is connected to numerous end users who are using edge devices in the IoT context. Data transfers may encounter traffic jams at the cloud side when a quantity of service or connection requests are concurrently received [8], [9]. In a real-world application scenario, this problem becomes more prominent for enterprises (e.g., the financial industry) that have to deal with a large number of simultaneous services within a time period [10].

Energy cost of cloud-computing platforms increases due to heavy workload communication [11]. This phenomenon weakens the advantage of using cloud-based applications in both working efficiency and energy-saving perspectives. The core limitation being faced by a cloud-IoT model is that a large-sized data transfer takes an excessive volume of computing resources and; consequently, expensive data transfers offset the benefits of centralized computing in clouds [12]. Partial energy costs consumed by network transfers are used for computing low-valued tasks, which is an unnecessary energy waste. Therefore, there is an urgent demand for exploring an effective approach with a real-time dynamic alternatives to alleviate heavy data transfers.

- *Keke Gai and Liehuang Zhu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {gaikeke, liehuangz}@bit.edu.cn.*
- *Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347 USA. E-mail: xqin@auburn.edu.*

Manuscript received 15 Aug. 2019; revised 22 Apr. 2020; accepted 2 May 2020. Date of publication 11 May 2020; date of current version 11 Mar. 2021. (Corresponding author: Keke Gai.) Recommended for acceptance by X. Cheng. Digital Object Identifier no. 10.1109/TC.2020.2993561

0018-9340 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

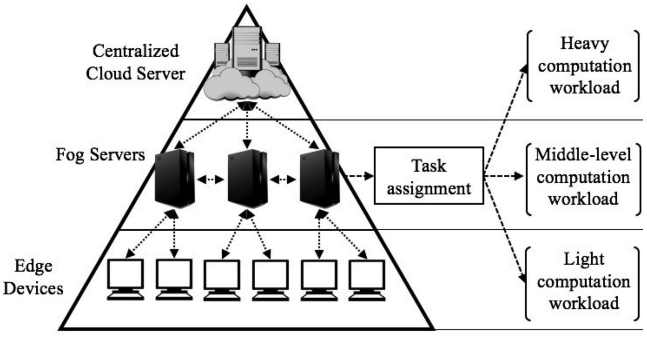


Fig. 1. High level architecture of the proposed model.

Next, the extreme demand also makes the significance of this work perceptible. Many modern enterprises have already adopted cloud services as well as an IoT-based connected operation environment. The expeditious growing scope of IoT introduces new issues of the energy supply, such as the energy shortage and imbalanced supply. Developing an advanced energy conservation approach delivering high computing performance and being compatible with existing systems is a foremost and principal issue for those users who deploy large-sized cloud-IoT solutions.

Fog computing is a potential solution to the aforementioned problem. In contrast to a cloud deployment, fog computing injects an additional layer between central cloud servers and end devices [13], [14], [15]. The additional layer deploys fog servers for the purpose of networking controls and governance, thereby further optimizing existing cloud systems. Furthermore, IoT offers the probabilities of improving the utilization of the devices by connecting devices throughout networks. A critical challenge is restricting the implementation of fog computing in optimizing computing resources. Many contemporary IoT systems in the industry involve a large number of devices in a wide scope. It is nontrivial for fog servers to efficiently dig out effectual task assignments that fully utilize available computing resources.

Most current active solutions in fog computing implement basic operations in order to reduce complexity. Fog servers typically receive and forward input tasks to certain pre-configured servers or edge devices, which are recorded in a table. It can neither effectively achieve a real-time optimization nor reach a high-level usage of all equipment. The task forwarding strategy is static in that some connected devices are encountering heavy workloads whereas others are idle. In addition, the advantage of a dynamic task assignment is observable but it faces a great challenge in strategy designs. On one side, creating optimal solutions generally requires a dramatically long time period due to its exponential growth in complexity. On the other side, a non-optimal solution is not guaranteed to be superior to the existing solutions.

It is arguably true that a feasible alternative is to design an approach to producing near-optimal solutions with an acceptable plan generation time. Our scheme is focused on two crucial facets, namely, task-assignment efficiency and energy-saving performance. Fig. 1 illustrates our proposed system architecture, in which three layers (i.e., centralized cloud servers, fog servers, and edge devices layers.) are responsible for computations. These layers shape a heterogeneous computing environment due to a diversity of computation performances of servers and devices. We consider all servers and edge devices with computation capabilities the computing units, which participate in data processing. Task forwarding are dynamically handled, depending on workload types.

To achieve a lower-level energy cost under a certain timing constraint, we propose a novel approach called the *Energy-aware Fog Resource Optimization* model or *EFRO*. The model constitutes three components, namely, cloud, fog, and edge layers. Each device or equipment in the model is considered a computation unit. In the model, a tuple 2-$\langle E_{ij}, \mathcal{T}_{ij}\rangle$ represents energy cost E and time consumption $\mathcal{T}$ of task T_i at computation unit C_j. The goal is to minimize total energy cost, $\sum E_i$, under a pre-defined timing constraint. The computation complexity should be exponentially growing when the number of input tasks or available heterogeneous computing units increases. With the model in place, we propose a heuristic algorithm to create near-optimal solutions in which overall performance of an entire system is considered.

The main contributions of this work are threefold:

1) This study addresses an important issue in the field of high-performance fog computing, which covers both execution efficiency and energy consumption. Our developed approach is conducive to producing near-optimal solutions in an instant manner, even though the constraints in multi-dimension raise system-design complexity.
2) The novel heuristic algorithm incorporates a Hill-climbing mechanism, which utilizes standardization and smart shift operations.
3) Our approach is a ubiquitous solution that is applicable in edge computing. The model can be transplanted to an edge computing system by treating edge devices as computation units.

The rest of this paper is organized by the following structure. First, Section 2 reviews relevant work in fog computing and its optimization mechanism in order to picture the research background. Next, Section 3 presents the key concepts and the proposed model as well as problem definitions. Furthermore, Section 4 further displays the core algorithms developed for the implementation of the our model. In addition, Section 5 demonstrates the proposed model by presenting partial experimental results. Finally, Section 6 draws conclusions of this work.

2 Related Work

This section focuses on reviewing challenge issues in energy-saving and resource management in fog computing as well as summarizing solutions investigated in recent studies.

The term *Fog Computing* (a.k.a., fog networking), derived from a gradual development of cloud computing. Some other terms used to be applied to describe a networking node that is similar to the functionality of a fog computing node, such as a Cloudlet [16], [17]. The concept of fog computing covered a wider service scope compared with the prior term, which involved most contemporary computing services, such as data storage, transmissions, networking controls and governances, system measurements, and security management [13], [18], [19]. For example, DECM [16] is an optimal solution powered by dynamic programming in

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

energy-saving scheduling scenarios of cloudlets MESF [20] is a greedy-based optimization algorithm that is derived from Round-Robin (RR) strategies [21]. The principal mechanism is offloading computing load from cloud servers to an augmented new fog server layer [22]. Our work expands this principle by launching an implementation to a broader scope. Fog server layer assigns tasks to three layers: the cloud-server, fog-server, and edge- device layers.

Despite an extended service scope explored by the previous fog computing research, there exit a few challenges restricting the implementation of fog computing in practice. One of the challenges lies in the problem of the resource allocation (a.k.a., task assignments), the complexity of which becomes more pronounced due to heterogeneous networks [23]. From the perspective of communications, the challenges exist in interactions between fog layers and *Data Service Operators* (DSOs) or *Authorized Data Service Subscribers* (ADSSs) [24]. The limitation generally is attached to the complication of protocol designs. Another challenge in this aspect is maintaining high reliability and adaptability in fog computing [25]. Some typical dimensions include declarative response time, programmable communications, and anti-interference memory management. Our work aims to further solidify the theoretical underpinnings for protocol designs from a viewpoint of the strategy generation for task allocations.

Moreover, fog computing is facing a challenge caused by bandwidth restrictions and energy supplies [26] from the angle of transfer capacity. Li *et al.* [27] presented a coding-size reduction method, which consists two types of coding, namely *minimum bandwidth codes* (MBC) and *minimum latency codes* (MLC), which are designed for distinct scenarios in distributed fog computing. However, reducing coding sizes is not a customized technique for fog computing. Similar work addressed developing techniques in reducing coding time [28] or reducing complexity of the coding [29], which is restricted by application scenarios. Data structures and coding units could impact the effect of the coding size reduction. In our study, we consider task response time at a computation unit given a fixed input without concerning the coding techniques.

Next, developing optimization algorithms was an alternative for lowering down negative impacts caused by the problem of the task allocation. For instance, Gu *et al.* [30] proposed a cost-efficiency approach to combining fog computing with *medical cyber-physical systems* or MCPS. Addressing the characteristics of MCPS, this approach - fueled by a greedy algorithm - only copes with limited constraint dimensions. The strategy generation time was efficient but the outputs still were sub-optimal solutions. Yang *et al.* [31] advocate for computation partitioning technique in resource allocation with an offline algorithm. A trade-off was made between efficiency and other computing resources (i.e., network and infrastructure) thanks to the involvement of parallel computing. It is worth noting that there are a handful of prior studies exploring optimization algorithms in practical ways (see, for example, deep reinforcement learning in strategy generations [32]). This type of research is one of the mainstreams in optimizing task allocations of fog computing [33], [34]. Unlike this study, our work aims to develop a practical approach to fulfilling both strategy−generation time and energy-saving requirements.

Furthermore, a raft of studies [35] exploited fog computing in a viewpoint of *intelligent transportation systems* or ITS. This route, being different from previous pure cloud-based solution [36], [37], involved edge units in ITS to participate in computations (e.g., vehicular computing). In order to adapt fog computing in a dynamic vehicular networks, a road-side cloud layer [38] could play a real-time networking controller role. A scheduling scheme could be constructed by assigning tasks to a shortest queue by predicting response times. Involving servers and edge devices from multiple layers introduces a few new obstacles in a real-time service scene, such as efficient response and integrity in heterogeneous computing. Zhou *et al.* [39] simplified the task allocation issue to a two-side matching problem in vehicular fog computing. The preference list was configurable by the virtue of the implementation of its pricing-based stable matching algorithm. Moltafet *et al.* [40] further formulated the resource allocation problem into an integer linear programming problem, which is applicable in the cloud-radio access networks. Similarly, another study [41] investigated the possibility of using hybrid methods, combining non-orthogonal multiple access with orthogonal frequency division multiplexing, to achieve energy-efficient resource allocations. This study lowered down the complexity by converting a non-linear programming issue to the tractable decoupled problem. We observe that the previous work emphasized output performance rather than a strategy-generation process. A continuously changing computation units in networks imposes an assumption that constructing a dynamic table for task assignments is adaptable. Some studies partially addressed the resource allocation issue by simplifying the problem (e.g., reducing complexity) [42]. Degrading complexity generally restricted the implementation as a list of specifications have to be excluded. Relaxing this assumption, our work pays heed to the implementation of a fixed table while aiming to partially solve the proposed NP-hard problem.

In addition, a few scholars attempted to optimize fog computing through physical cases. Hu *et al.* [43] evaluated the adoptability of using fog computing to process face identifications, which typically require high-performance computing resources in data processing and data transfers. The proposed framework is a static mechanism where each input task is processed at a fog computing node and outcomes are synchronously sent to end users and cloud servers. Chen *et al.* [44] delved into short packet communications in wireless-powered IoT network to power up transmission efficiency. This work was focused on transmission time while overlooking the other constraints. In contrast to the above solutions, our scheme is a generic technique that can be readily adopted in a raft of application domains. The transmission efficiency issue in media handling services was also addressed and reported in [45], in which the optimization case was formulated into an ILP problem. The solutions presented in [44], [45] only offer optimization results with limited specifications due to the degrading complexity. In contrast to the above solutions, our research has a distinct focus in that ours is a generic technique that can be readily adopted in various application domains. In other words, our system facilitates a ubiquitous approach concentrating on a diversity of computing tasks in an abstract way instead of a concrete manner.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

In a nutshell, even though a handful of previous studies have investigated optimization methods in cloud/fog/edge resource allocation, the high-complexity allocation problem has been rarely addressed without degrading complexity. It is a conventional wisdom to convert the allocation problem to an ILP problem, where the implementation is restricted by the limited condition. However, it is unusual to seamlessly meet the conditional assumption in practice, which weakens the adaptability of the existing techniques in the real-world heterogeneous fog context. In contrast to the prior work, our proposed solution is an effective extension in the realm of fog computing. All computing devices are treated as computation units throughout the three layers. Our solution is reliant on fixed tables stored in fog servers. Our heuristic algorithm is adept at processing optimization operations for task allocations.

3 Concepts and the Proposed Model

3.1 Problem Definition

The primary issue addressed in this work is making task-allocation decisions in the context of heterogeneous fog computing. The diversity of computation units from the cloud, fog, and edge layers spur challenges in the resource allocation techniques. The formulation of the heterogeneous environment derives from varied characteristics of involved entities (e.g., cloud, fog, and edge computing). More broadly, cloud servers can afford to deal with heavy workloads, which may cause burden of network bandwidth; edge devices with limited computation capabilities require less network resources; fog servers exhibit a stronger capability in interconnections among entities (e.g., task allocation, temporary storage). Therefore, it is appealing to design novel strategies to cope with task allocations in various heterogeneous environments (input tasks and workload costs at cloud/edge). The communication cost is priced into the time consumption, so we directly incorporated the delay measures into the time consumption. The time consumption of the a task on the same type of servers might be different. After modeling heterogeneous fog computing platforms, we articulate the problem statement formally presented in Definition 3.1.

Definition 3.1. The Time-Constraint Energy Minimum Problem (TCEMP): *Inputs include a set I of tasks $\{T_i\}$ ($i \in N$, $0 < i \leqslant I$), a set J of available computing units $\{C_j\}$ ($j \in N$, $0 < j \leqslant J$), a pre-stored table TB recording all tuples of energy costs and time consumptions $\langle E_{ij}, \mathcal{T}_{ij}\rangle$ and the communication costs $M_{j,j'}$ between any pair of two computing units C_j and $C_{j'}$, a timing constraint $\mathcal{T}^c$, $J \geqslant I$.*

Output is a mapping table P prescribing task assignments.

The proposed TCEMP problem is to search near-optimal solution to minimize overall energy cost (E_{total}) under timing constraint $\mathcal{T}^c$.

TCEMP is mathematically expressed in Eq. (1)

$$E_{total} = \mathrm{MIN}\Big[\sum E\Big] \text{ and } \mathcal{T}^c \geqslant \sum \mathcal{T}. \tag{1}$$

In Eq. (1), *MIN*$[\cdot]$ is a function that elects the minimum value from an input. $\sum E$ refers to the total energy cost. $\sum \mathcal{T}$ represents the total execution time length (a.k.a., makespan). In this problem, we consider the total execution time summing up all time lengths rather than a overlapped time with parallel computations. The values of $\sum E$ and $\sum \mathcal{T}$ are obtained from Eqs. (2) and (3), where $s(\cdot)$ is a determinative function and its output value determines the objective of a computing unit; $c(\cdot)$ is a boolean function signifying whether a computing unit is selected. Additionally, $s(i) = 1$ means that the computing unit is assigned to a task; $s(i) = 0$ means that the computing unit is unassigned.

$$\begin{aligned}\sum E &= \sum_{a=1}^{J} \sum_{\substack{s(i)=a,\\ c(i)=1,\\ i=1}}^{I} E_i \\ &= \sum_{\substack{s(i)=1,\\ c(i)=1,\\ i=1}}^{I} E_i + \sum_{\substack{s(i)=2,\\ c(i)=1,\\ i=1}}^{I} E_i + \ldots + \sum_{\substack{s(i)=J,\\ c(i)=1,\\ i=1}}^{I} E_i.\end{aligned} \tag{2}$$

$$\begin{aligned}\sum \mathcal{T} &= \sum_{a=1}^{J} \sum_{\substack{s(i)=a,\\ c(i)=1,\\ i=1}}^{I} \mathcal{T}_i \\ &= \sum_{\substack{s(i)=1,\\ c(i)=1,\\ i=1}}^{I} \mathcal{T}_i + \sum_{\substack{s(i)=2,\\ c(i)=1,\\ i=1}}^{I} \mathcal{T}_i + \ldots + \sum_{\substack{s(i)=J,\\ c(i)=1,\\ i=1}}^{I} \mathcal{T}_i.\end{aligned} \tag{3}$$

We intend to devise near-optimal solutions, because the *TCEMP* problem is NP hard. Producing an optimal solution to the proposed problem is always time consuming regardless of any designed algorithm [46]. We propose Theorem 3.1 to simplify the complexity of the TCEMP problem. The problem stated in Theorem 3.1 merely digs deep into energy cost without examining move cost and; therefore, the complexity of TCEMP is higher than the problem described in Theorem 3.1. In what follows, we present Theorem 3.1 along with its brief proof.

Theorem 3.1. *$\exists$ a set of tasks $\{T_i\}$ with I tasks and a set of computing units $\{C_j\}$ with J tasks, where $i \in N$, $j \in N$, $0 < i \leqslant I$, $0 < j \leqslant J$, $J \geqslant I$. $\forall T_i \in \{T_i\}$, there is a unique tuple at each $C_j \in \{C_j\}$, which is $\langle E_{ij}, \mathcal{T}_{ij}\rangle$. Then, finding out an optimal solution to minimizing the total $\sum E_i$ under a timing constraint $\mathcal{T}^c$ is an NP-hard problem.*

Proof. Suppose there are n groups of input tasks and m groups of available computing units, and a pre-stored table recording all tuples of energy costs and time consumptions $\langle E_{ij}, \mathcal{T}_{ij}\rangle$ is given. A traversing search is essential to obtain an optimal solution that minimizes the total energy cost by comparing all combination $\langle E_{ij}, \mathcal{T}_{ij}\rangle$ considering the timing constraint. Computing all potential combinations requires P! Times. Based on this knowledge, the heterogeneous fog computing platform can be constructed into several backpacks with different capacities. The capacity of the backpack refers to the capacity of each computing platform, such as the computing cost under each time constraint. The input task is items that need to be put into the backpack. Different items have different volumes and values, for example, different tasks consume different energy and time. In addition, our goal is to generate the minimum cost through the permutation of p. Therefore, the TCEMP problem can be reduced to

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

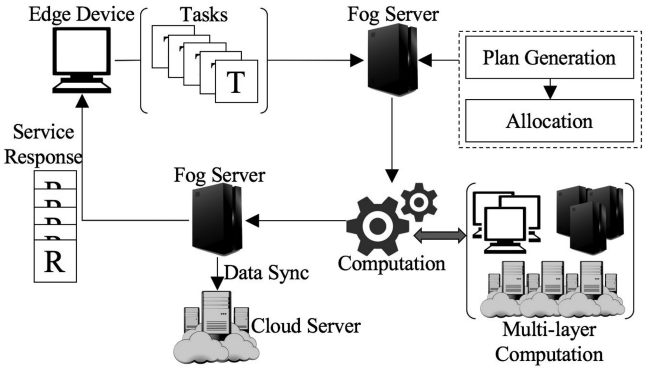


Fig. 2. The workflow of the proposed EFRO model.

Knapsack Problem (KP), a combined NPhard problem. It can be proved that TCEMP is also an NP-hard problem, thanks to BBP $\geqslant_p$ TCEMP. □

The above proof indicates that TCEMP is equally provable to be an NP-hard problem.

3.2 System Design

Now we shed some light on the system design of our proposed approach.

3.2.1 Overview of the Proposed Model

The operating principle of our system design is to allocate tasks to appropriate computation units. In line with the architecture shown in Fig. 1, the basic idea is using centralized cloud servers to process computing-intensive workloads and relying on fog servers and edge devices to process middle- and light-weighted workloads, respectively. In addition, our model utilizes the quantitative criterion to determine workload levels and computation units' capacities. The criterion of computation energy costs are stored at the fog server layer. In our model, each fog server has a pre-stored table that maps energy cost for certain tasks at connected computation units. Some energy-cost values in the table might be same or similar because of the same hardware and networking configuration or a homogeneous computing environment [47]. Thus, the pre-stored table categorizes computation units at each layer.

Categorizing computation resources is of paramount importance for increasing allocation efficiency and avoiding computational overhead caused by slight differences of the energy cost between computation units. Checking availability of each category of computation units only needs call the table to identify the number of the unit rather than searching all connected units. Therefore, each category of the computation unit in the pre-stored table can be represented by a 3-tuple: $\langle E, T, N\rangle$. Within the tuple, E refers to energy cost, T denotes the corresponding task type, and N is the available amount of this category of the computation unit. Meanwhile, each computation unit is represented by a 4-tuple (i.e., $\langle E, \mathcal{T}, T, C\rangle$), where E and T are the same as those in $\langle E, T, N\rangle$; C is a label to the computation unit; $\mathcal{T}$ is the time length estimate for task T at computation unit C. This approach provisions the input retrievals for task allocations.

In the proposed model's high-level workflow illustrated in Fig. 2, the edge device collects a sequence of incoming tasks delivered to a nearby fog server. Incoming tasks are modeled as a set $\{T_i\}$, where $i \in \mathbb{N}$, $i \in [1, I]$. In order to process task allocations, the fog server retrieves the header of each task and creates a cost mapping table based on pre-stored reference dataset. A plan for task allocations is derived from the mapping table, thereby determining final allocations. The workflow is at the heart of our model, in which a handful of algorithms are proposed to complete the task-allocation process. Creating a task allocation plan is complicated, because computations take place at multi-layer devices like edge devices, fog servers, and remote cloud servers. The output of an allocation operation is a set of computation unit addresses to which tasks are assigned.

Finally, tasks are sent to the corresponding computation units followed by the plan; results are returned to the fog server after computations are accomplished. On one side, the results are delivered back to the edge devices as service responses. On the other side, data are synchronized at the remote cloud server or storage-oriented fog server near the edge device. It completes an operation cycle for computation tasks up to this point. We mainly discuss the task allocation procedure; and data synchronizations and data retrievals will be addressed in a future study.

3.2.2 Task Allocation Procedures

At the center of our approach, the task allocation mainly embraces two procedures. The first procedure is to create a task allocation plan with a priority considering the execution time in order to ensure that an output solution complies with timing constraint $\mathcal{T}^c$. The purpose of this procedure is to guarantee all tasks are completed within the timing constraint. The priority principle follows two assumptions from empirical studies [48], [49], [50], [51], which are: 1) the energy cost has a near-linear positive relationship with the length of the implementation time for the same incoming task with the same hardware configuration; 2) the energy cost has a non-linear relationship with the length of the implementation time for the same incoming task when hardware configurations vary. Hence, the fog server should acquire the execution time parameters to estimate the total estimate time of the allocation plan at this stage. The outcome of this procedure is a near-optimal solution to minimizing the total execution time. Energy cost is not a concern in this procedure.

Creating an allocation plan depends on the implementation of our core algorithms. The core algorithm in our model applies the Hill-climbing mechanism, which transfers a sub-optimal solution into a near-optimal solution. Improvements are made by a few adjustment operations on the sub-optimal solution that is yielded from a greedy algorithm. We implement a greedy algorithm aiming to result in low computation complexity. The expected execution time length for creating a sub-optimal solution is short, since its output is only an intermediate variable. We call this operation as a *Smart Shifting* operation that consists of searching and transposing manipulations. We also refer to this operation as a shifting operation, the description of which is presented in Section 3.2.3.

The goal of the second procedure is to cut back total energy cost. The mapping table updated by the first procedure is an input of this procedure. The first step is to transfer the cost mapping table into an energy-oriented version according to an allocation plan.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

The second step is implementing shifting operations. Distinguished from the first step, this step checks time length after each shift operation in order to guarantee that the total execution time is within the timing constraint. Finally, the final allocation plan is constructed either when no shifting operation is needed or reaching the timing constraint. The key algorithms supporting shifting operations is delineated in Section 4.

3.2.3 Shifting Operation

A shifting operation is comprised of two major procedures. The first procedure deploys a greedy algorithm to establish a sub-optimal solution. A standardization table is originated to identify the differences among various computation units selections. The other procedure is the task allocation's vital component - a shifting operation over a sub-optimal solution.

The purpose of the shifting operation is to transpose a number of computation units selections in order to formulate re-allocations to slash energy cost. The shifting-operation design is inspired by a searching-transposing manipulation. A searching manipulation searches all negative values in a standardization table and assesses whether a transposition can improve the solution. A transposition manipulation is carried out once a matching condition is found. The search-transposing manipulations are detailed in Section 4.2.

4 Algorithms

We have developed a number of algorithms to support the implementation of our model. The core algorithm is designed to shift task allocations for boosting energy-saving performance and working efficiency, which is a centralized algorithm running at the fog server layer. The implementation of the proposed algorithm facilitates fog servers to allocate tasks toward cloud/edge sides. We are inspired by the cliff climbing method to propose our algorithms, which are adroit at transferring a non-optimal strategy to a near-optimal solution. The transferring process - a core operation - is governed by the algorithms elaborated in this section. There are stark difference between our algorithms and the existing schemes, because our algorithms address the drawbacks of the greedy algorithm-based solutions that fail in offering optimal task allocation decisions. The mathematical proofs are discussed in this section.

Moreover, We have versatile types of tasks (e.g., light and heavy workloads). The investigated systems embrace various servers such as cloud tier servers, fog tier servers and edge devices. When the number of tasks and the number of servers are not large, we manage two tables with fixed values of energy consumption and delay through testing. In case of an enormous number of tasks and servers, we obtain the two tables in the following approach. Through testing different types of tasks and different types of servers, we create two tables that float up and down in accordance with a fixed value.

In reality, the arrivals of new tasks and the completions of old tasks are constantly changing, which in turn makes server availability dynamically updated. Although the task allocation decision for the next time slot is highly relevant and affected by the current allocation, we can encapsulate this dynamic process into a group by obtaining an available server pool and newly arrived tasks for the current time slot problems in which servers have fixed power consumption and tasks' arrival rates are given in a priori. As such, we focus on performance optimization over a series of discrete time slots. Therefore, there is no need to wait for all tasks to be completed in the current time slot before assigning newly arrived tasks. Our task scheduling scheme performs in an online fashion to respond to dynamically changing networks.

4.1 Standardization Table Creation (STC) Algorithm

The expectation of designing *Standardization Table Creation* (STC) algorithm is to create an intermediate table showing potential better assignments. Its basic idea applies the operation of the standardization to formulate the cost mapping table into a comparable manner. The mechanism is subtracting the energy cost value of the selected computation unit from energy cost values for the same input tasks at all available computation units in the cost mapping table.

The inputs of the STC algorithm include a cost mapping table TB and a set $\{ \underset{c(i)=1}{X_{ij}} \}$ that represents a sub-optimal solution of the task allocation. We use "X" to represent the allocation set, since the objective can be either energy (E) or time ($\mathcal{T}$). Specifically, set $\{ \underset{c(i)=1}{X_{ij}} \}$ represents the first step of our proposed scheme; thus, its status is always dynamically changing. Our proposed algorithm adopts a cliff climbing approach in a way that the first step is to kick in a non-optimal/sub-optimal strategy. The intent of Algorithm 1 is to convert a dynamic input to the format that can be further processed by our optimization algorithms.

The mapping table TB can be represented by a matrix $\mathbf{TB} = (E_{ij}) \in \mathbb{R}^{I \times J}$, when there are I input tasks and J available computation units in total. The set $\{X_{ij}\}$ is created by operating a greedy algorithm [52], [53] on the cost table. The output of the STC algorithm is a new mapping table, ST, which uses "0"s to represent the selected computation units for each task. Let E_i^s be the selected computation unit for the incoming task T_i; thus, there exists $X_i^s \leftarrow \{ \underset{c(i)=1}{X_{ij}} \}$. In line with the notations in Section 3, the boolean function $c(\cdot)$ represents the state of the computation unit selection. The output mapping table can be represented by a $I \times J$ dimensions matrix and its mathematical expression is shown in Eq. (4)

$$\mathbf{ST} = (X_{ij} - X_i^s) \in \mathbb{R}^{I \times J}. \tag{4}$$

We provide pseudo codes of the STC algorithm in Algorithm 1. A few vital phrases of the STC algorithm are given in the followings:

1) Read the input table and a set for the original assignment plan. Initialize a new table for the output.
2) Enter a FOR loop. Read one element from the assignment set at one time and operate subtractions on all elements at the same row. The FOR loop ends when the subtraction operations have been operated on all elements in the table.
3) Update the output table and return it.

The time complexity of this algorithm is $T(n) = O(IJ)$ based on the size of the standardization table. Assume that the table have dimensions $I \times J$, such that each element in the table is only operated once. Thus, $I \times J$ computations is

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

required to create an ST table. The output of STC algorithm will be used in our smart shift operations.

Algorithm 1. Standardization Table Creation Algorithm

Require: TB, $\{ X_{ij} \}_{c(i)=1}$
Ensure: ST
1: Input the original table TB and read $\{ X_{ij} \}_{c(i)=1}$
2: Initialize a new table $ST \leftarrow \emptyset$ with I rows and J columns
3: /* Each element in ST: X'_{ij} */
4: **for** each row in TB **do**
5: Read $\{ X_{ij} \}_{c(i)=1}$ for the ith row
6: $X_i^s \leftarrow X_{ij}{}_{c(i)=1}$
7: **for** all elements at the ith row **do**
8: $X'_{ij} \leftarrow (X_{ij} - X_i^s)$
9: Add X'_{ij} to ST
10: **end for**
11: **end for**
12: **return** ST

4.2 Smart Shift Algorithm

4.2.1 Theoretical Synthesis for Smart Shift

The *Smart Shift* (SmS) algorithm is the core algorithm in our EFRO model, by which a sub-optimal solution is improved to become a near-optimal solution. It is the main algorithm for operating the shifting operation to reduce the total time length or energy cost by transposing the computation unit selections in edge computing. The intent of this operation is checking the reason for causing a non-optimal solution obtained from a greedy algorithm. One reason is that, for the same task, the selected computation unit could be worse than some other unselected computation units that are already taken by other tasks, because a greedy algorithm creates a solution by considering the priority. Incoming tasks are read in a sequence so that the taken computation unit maybe more suitable for the succeeding tasks. Therefore, the goal of SmS algorithm is to make a few shifts to avoid the situation above if applicable.

Moreover, when processing the energy reduction, the inputs of the SmS algorithm include a mapping table that is the output of the STC algorithm, represented as ST, as well as a set $\{ E_{ij} \}_{c(i)=1}$. Smart shifts will make changes on the set $\{ E_{ij} \}_{c(i)=1}$ by checking and transposing a number of selections. The output is an updated set $\{ E^u_{ij} \}_{c(i)=1}$ that represents a task assignment plan. Similarly, when reducing the length of the time execution, inputs for SmS algorithm include ST and $\{ \mathcal{T}_{ij} \}_{c(i)=1}$ and the output set is $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$.

Next, we use a term *cost* to describe both energy and time in the explanation of the mechanism. Crucial phrases are presented as follows:

Phrase I. In order to expressively explain the proposed algorithm, we created function $f(\cdot)$ representing the status in the standardization table. The input of function $f(\cdot)$ is an original data in the original table, and the output of function $f(\cdot)$ is the corresponding data in the standardization table. The calculation process of function $f(\cdot)$ resembles the execution of Algorithm 1; the calculation of function 1 is articulated in details in the explanation of Algorithm 1. For example, $f(T_h)_{s(i)=p,c(i)=1}$ represents a cost value for the task T_h at the selected computation unit C_p; $f(T_h)_{s(i)=q,c(i)=0}$ represents a value for the task T at the unselected computation unit C_q. According to the method given in Algorithm 1, there exists a $f(T_h)_{s(i)=p,c(i)=1} = 0$. We use the expression $A >^? B$ to represent an operation that assesses whether the value of A is bigger than B.

Phrase II. Based on this setting, the searching process is to seek any situation of $f(T_h)_{s(i)=p,c(i)=1} > f(T_h)_{s(i)=q,c(i)=0}$. The following operation shown in Eq. (5) will be processed when the situation is found

$$f(T_h)_{s(i)=q,c(i)=0} + f(T_g)_{s(i)=p} <^? 0. \tag{5}$$

$f(T_g)_{s(i)=p}$ refers to another task T_g and its status of the computation unit selection in the standardization table. A shift operation will be implemented when $f(T_h)_{s(i)=q,c(i)=0} + f(T_g)_{s(i)=p} < 0$ is true.

Phrase III. The shift operation will re-select computation units by exchanging the units for tasks T_h and T_g. Task T_h will be assigned to the computation unit C_q and task T_g will be assigned to C_p. Thus, values of $f(T_h)$ and $f(T_g)$ will be re-calculated by using the method in Algorithm 1. The updated selected computation units for tasks T_h and T_g are $f(T_h)_{s(i)=q,c(i)=1}$ and $f(T_g)_{s(i)=p,c(i)=1}$. The shift operation will not end until there is no condition mentioned in Phrase II.

In addition, we present a theorem to ensure the correctness of the shifting operation. The core idea of the shifting operation is based on a fact that each transposing 2 tasks will reduce the total energy cost and the decrease is only related to these 2 transposed tasks. Therefore, the problem can be simplified into a 2-row matrix problem from m-row matrix problem. Next, each transposition only involves two computation units in that two columns in the mapping table participate in the shifting operation. It means that the problem can be further formulated into a problem represented by a 2×2 matrix. Theorem 4.1 provides the presentation of the proposed theorem.

Theorem 4.1. *$\exists$ a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{2\times2}$. Implement a standardization operation over the matrix and ensure only one element at each column can be a subtrahend. Thus, a new matrix can be obtained, which is denoted by $\mathbf{B} = (b_{ij}) \in \mathbb{R}^{2\times2}$. The operation is denoted by $\mathbf{B} \overleftarrow{s} \mathbf{A}$. There exist 2 situations for standardization operations so that matrix $\mathbf{B}$ has two manners:*

$$\mathbf{B_1} = \begin{bmatrix} 0 & b_{12} \\ b_{21} & 0 \end{bmatrix} \text{ or } \mathbf{B_2} = \begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix}.$$

Therefore,
$(a_{11} + a_{22}) > (a_{12} + a_{21})$ is true, if $\exists\, b_{12} + b_{21} < 0$ in $\mathbf{B_1}$.
$(a_{12} + a_{21}) > (a_{11} + a_{22})$ is true, if $\exists\, (b_{11} + b_{22}) < 0$ in $\mathbf{B_2}$.

Moreover, we provide a brief proof for Theorem 4.1.

Proof. According to the condition given in Theorem 4.1, matrix $\mathbf{B}$ two potential matrices forms, $\mathbf{B_1}$ and $\mathbf{B_2}$, after a standardization operation.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

In $\mathbf{B_1}$, a_{11} and a_{22} are selected as subtrahends such that $\mathbf{B_1} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} - a_{11} & a_{12} - a_{11} \\ a_{21} - a_{22} & a_{22} - a_{22} \end{bmatrix}$.

Since $b_{12} + b_{21} < 0$,

$\Rightarrow a_{12} - a_{11} + a_{21} - a_{22} < 0$

$\Rightarrow a_{12} + a_{21} < a_{11} + a_{22}$, which is $(a_{11} + a_{22}) > (a_{12} + a_{21})$.

Thus, the first conclusion in Theorem 4.1 is correct. Using the same method also can infer a conclusion for $\mathbf{B_2}$:

Since $(b_{11} + b_{22}) < 0$,

$\Rightarrow a_{11} - a_{12} + a_{22} - a_{21} < 0$

$\Rightarrow a_{11} + a_{22} < a_{12} + a_{21}$, which is $(a_{12} + a_{21}) > (a_{11} + a_{22})$.

Therefore, Theorem 4.1 is proved. □

4.2.2 Algorithm Design

The SmS algorithm has two versions that are designed for dealing with reductions of the time and energy, respectively. Two versions of SmS algorithms are called *Smart Shift Time* (SmS-T) and *Smart Shift Energy* (SmS-E) algorithms. This section concentrates on describing the implementation and design of these two algorithms.

Algorithm 2. Smart Shift Time Algorithm

Require: ST, $\{ \mathcal{T}_{ij} \}$

Ensure: $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$

1: Input ST, $\{ \mathcal{T}_{ij} \}_{c(i)=1}$; initialize $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$
2: /*Original ST derives from Algorithm 1 and will be updated during the shift operations*/
3: **while** $\forall$ rows in ST, $f(T_h)_{s(i)=p,c(i)=1} > f(T_h)_{s(i)=q,c(i)=0}$ **do**
4: Process the comparison $f(T_h)_{s(i)=q,c(i)=0}$, $f(T_g)_{s(i)=p}$
5: **if** There exists $f(T_h)_{s(i)=q,c(i)=0} + f(T_g)_{s(i)=p} < 0$ **then**
6: Transpose the computation unit allocations
7: Allocate new allocation to $f(T_h)_{s(i)=q,c(i)=1}$, $f(T_g)_{s(i)=p,c(i)=1}$
8: Update table ST
9: **end if**
10: /*Searching will not be ended until no matching condition is found.*/
11: **end while**
12: Create a $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$ deriving from the final ST.
13: **return** $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$

First, SmS-T algorithm is developed for creating an intermediate allocation plan that is efficient enough to complete all tasks. The main input of SmS-T algorithm includes a standardization table (ST) that contains an initial task allocation plan. This allocation plan is created by a greedy algorithm [53], which uses the time length as the priority without concerning the energy cost. The output of this algorithm is an updated allocation plan. Algorithm 2 presents the pseudo codes of TAS algorithm in order to further explain its mechanism. Main steps of Algorithm 2 include:

1) Read the original standardization table (ST) with the initial task allocation plan.
2) Search negative values at each row of ST and judge whether a shift operation can reduce total energy cost. The implementation mainly depends on the operations mentioned in Phrase II. A shift operation will be processed once a matching condition is found, such that two tasks transposes their computation unit selections.
3) Once the searching and transposing operations over the original ST table are complete, update ST according to new task allocation plans so that a new ST' is created.
4) Repeat the shift operations through a searching-transposing process and create a new ST table based on new transpositions.
5) Keep the operations until no more matching condition can be found. Output the final task allocation plan deriving from the final ST table.

We present a brief time complexity analysis of Algorithm 2. The analysis is based on two situations, namely, the best and the worst situations. Let the standardization table be a $I \times J$ matrix. The best case for Algorithm 2 is that there is no matching condition found in the first searching such that no shift operation is needed. The time complexity is $T(n) = O(IJ)$ in this case. Meanwhile, the worst case matches at least three conditions. The first condition is that the matrix of ST has the maximum amount of negative values. The maximum amount is $(IJ - J)/2$ in a matrix with dimensions $I \times J$. The next condition is the energy cost difference between the sub-optimal solution deriving from the greedy algorithm and the optimal solution reaches the maximum value. Let p be an energy cost value obtained from an optimal solution and q be an energy cost value obtained from a greedy algorithm that generally is a sub-optimal solution. Thus, the difference will be $|p - q|$. Finally, the last condition is that each searching-transposing operation only reduces one unit energy consumption. It means that the number of shifting operations will be $|p - q|$ to reach the optimal solution. Based on the conditions above, the total amount of the computation for the worst case is $|p - q| \times (IJ - J)/2$. Therefore, the time complexity of the worst case is $T(n) = O(|p - q|IJ)$.

Next, SmS-E algorithm is the other version of SmS algorithm, which is designed to minimize the total energy cost within the timing constraint. The smart shifting operation also is the core of this algorithm. As mentioned in the prior section, SmS-E needs a manipulation of timing check that makes it different from SmS-T. The checking manipulation takes place after each switch operation. The inputs of SmS-E algorithm include an energy version standardization table (ST^e) that contains an allocation plan constructed by Algorithm 2, the time mapping table $\mathcal{T}_{ij}$, and a timing constraint $\mathcal{T}^c$. The output of SmS-E algorithm is the final allocation plan. Algorithm 3 presents the pseudo codes of SmS-E algorithm.

Furthermore, as shown in the presentation of the pseudo codes, the mechanism of the smart shifting operation in SmS-E is as the same as it is in SmS-T. However, additional judgments are required due to the timing constraint. Once the searching manipulation finds a matching condition, it shall update the allocation time table and examine whether the total execution time is longer than the timing constraint. A transposing manipulation is allowed only when the total execution time is no longer than the timing constraint, $S_{\mathcal{T}} \leqslant \mathcal{T}^c$. Otherwise, the shift operation is denied. The standardization table is updated after each shifting operation.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

The condition for ending loops is that no qualified shifting situations can be found.

Finally, Algorithm 3 has the same time complexity as Algorithm 2. The next section presents an algorithm for manipulating the whole process of the allocation plan generation.

Algorithm 3. Smart Shift Energy Algorithm

Require: ST^e, $\{ E_{ij} \}$, $\{ \mathcal{T}^u_{ij} \}$, $\mathcal{T}_{ij}$, $\mathcal{T}^c$
Ensure: $\{ E^u_{ij} \}_{c(i)=1}$ $_{c(i)=1}$

1: Read ST^e, $\{ E_{ij} \}_{c(i)=1}$, $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$, $\mathcal{T}^c$
2: /*Apply Algorithm 1 on $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$ to construct $\{ E_{ij} \}_{c(i)=1}$*/
3: Initialize a variable calculating the sum of time: $S_{\mathcal{T}}$
4: Assign value $S_{\mathcal{T}}$ based on $\{ \mathcal{T}^u_{ij} \}_{c(i)=1}$, $\mathcal{T}_{ij}$
5: **while** $\forall$ rows in ST, $f(T_h)_{s(i)=p,c(i)=1} > f(T_h)_{s(i)=q,c(i)=0}$ **do**
6: Process the comparison $f(T_h)_{s(i)=q,c(i)=0}$, $f(T_g)_{s(i)=p}$
7: **if** There exists $f(T_h)_{s(i)=q,c(i)=0} + f(T_g)_{s(i)=p} < 0$ **then**
8: Transpose the computation unit allocations
9: Allocate new allocation to $f(T_h)_{s(i)=q,c(i)=1}$, $f(T_g)_{s(i)=p,c(i)=1}$
10: **if** $S_{\mathcal{T}} \leqslant \mathcal{T}^c$ after this shift operation **then**
11: Allow shift operation.
12: **else if** $S_{\mathcal{T}} > \mathcal{T}^c$ after this shift operation **then**
13: Deny shift operation
14: **end if**
15: Update table ST
16: **end if**
17: /*Searching will not be ended until no matching condition is found.*/
18: **end while**
19: Create a $\{ E^u_{ij} \}_{c(i)=1}$ deriving from the final ST^e.
20: **return** $\{ E^u_{ij} \}_{c(i)=1}$

4.3 Smart-Shifting Min-Min Algorithm

The *Smart-Shifting Min-Min* (SmS-MM) algorithm provides a holistic control for constructing the task allocation plan. The literature of the algorithm implies that there are two critical components in this algorithm, which are smart shifting and min-min manipulations. The smart shifting is demonstrated in Sections 4.2.1 and 4.2.2. In addition, *Min-Min* is describing an optimization process consisting of two procedures for minimizing two objectives in sequence. Our algorithm successively minimizes the total execution time and the total energy cost. Fig. 3 illustrates a concise diagram of the SmS-MM algorithm with a few tables and main algorithms or activities during the transfer. The first *Min* procedure applies SmS-T algorithm and the second *Min* procedure applies SmS-E algorithm.

The inputs of SmS-MM algorithm includes a set of $\langle E, \mathcal{T}, T, C \rangle$ that can be retrieved from the pre-stored table at the fog server. Its output is a set of selected computation units for constructing the task allocation. We present the pseudo codes of SmS-MM algorithm in Algorithm 4, which is aligned with the diagram shown in Fig. 3.

The next section will presents a simple example of implementing the proposed approach.

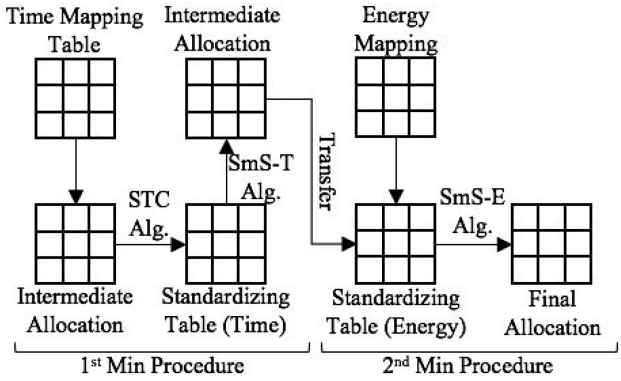


Fig. 3. Diagram of the Smart-Shifting Min-Min algorithm.

Algorithm 4. Smart-Shifting Min-Min Algorithm

Require: $\langle E, \mathcal{T}, T, C \rangle$
Ensure: $\{ E^u_{ij} \}_{c(i)=1}$

1: Input $\langle E, \mathcal{T}, T, C \rangle$ from pre-stored table, based on tasks
2: Construct a time mapping table
3: Create an intermediate allocation plan by using a greedy algorithm. Apply STC algorithm (Algorithm 1) to the intermediate allocation plan and the time mapping table to create a time version standardizing table.
4: Implement SmS-T algorithm (Algorithm 2) to construct a new intermediate allocation plan.
5: Read the energy mapping table. Apply STC algorithm (Algorithm 1) to the energy mapping table and the intermediate allocation plan to create an energy version standardizing table.
6: Implement SmS-E algorithm (Algorithm 3) to construct the final allocation plan $\{ E^u_{ij} \}_{c(i)=1}$ and return the plan set.

5 Experiment and the Results

5.1 Experimental Configuration

We conduct a series of experimental evaluations in order to quantitatively assess the performance of our *EFRO* approach. We implement a simulated fog computing environment on a server equipped with an Intel Core i5-7200U CPU, with a 4.0 GB main memory. We used our own simulator as a servers pool to complete the evaluation, which is done by Java development. In our extensive experiments, the network settings are configured in in the developed simulator, which can be tuned by various performance parameters (i.e., time, cost). In doing so, a diversity of cloud/fog server options are elected to delivery varied energy costs and time consumption. We configure the experimental settings to comprehensively emulate real-world scenarios in fog/edge computing. In doing so, we simulate the processing delay caused by a wide range of workload conditions and resource allocations in real-world situations and the network delay caused by tasks, users and processor locations. Data collections are based on 1×10^3 rounds of tests in each setting. We configure the energy consumption using a random value that follows a normal distribution, thereby resembling the real-world scenarios where workload allocations in different situations result in various energy costs for each server at each time slot. The extensive

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Experiment Settings Configuration

Settings	1-1	1-2	1-3	1-4	1-5	1-6	1-7
Task #	10	20	30	40	50	75	100
Settings	2-1(a)	2-1(b)	2-1(c)	2-1(d)	2-1(e)	2-1 (f)	2-1(g)
Task #	20	20	20	20	20	20	20
C_i Type #	4	5	6	7	8	10	20
Settings	2-2	2-3	2-4	2-5	2-6	2-7	-
Task #	30	50	50	50	100	100	-
C_i Type #	8	10	20	50	10	60	-

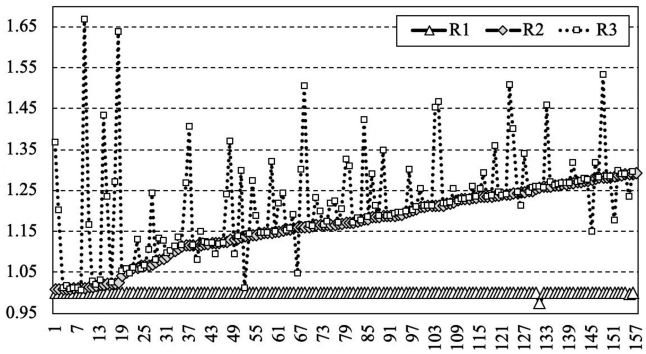


Fig. 4. Performance comparison in terms of R values under Setting 1-1.

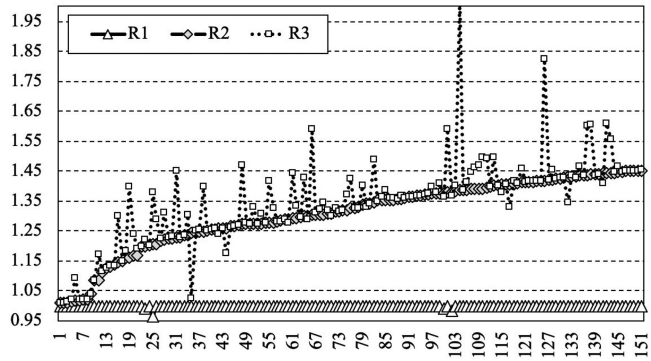


Fig. 5. Performance comparison in terms of R values under Setting 1-2.

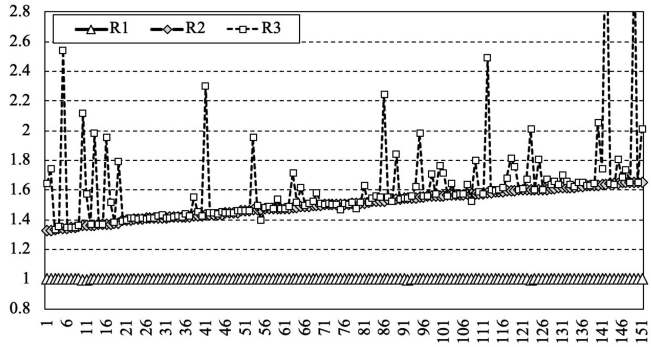


Fig. 6. Performance comparison in terms of R values under Setting 1-5.

experiments are focused on examining the performance of energy conservation and investigating the time length of each plan generation. To configure different experimental settings, we setup two variables - the number of incoming tasks and the categories of computation units. As discussed in the prior sections, computation units in the same category have the same energy-cost and time-consumption profiles.

In order to simulate real-world application environment, we adjusted the maximum number of nodes. Although the dataset used in the experiments is not a big set, the experimental results could sufficiently demonstrate that our algorithm is conducive to making nearly optimal decisions when resources are managed in a fog computing environment. The experimental results show that our algorithm orchestrates energy and time consumption in a holistic way.

In the first series of experiments (i.e., Setting 1), we compare the performance of EFRO and other alternative approaches. We fix the available computation-unit categories in setting 1, which includes seven sub-settings (i.e., settings 1-1 to 1-7). The main objective of this group of experiments is to study the impacts brought by the number of incoming tasks when the available computation resources are in a static status in a fog computing environment. Table 1 summarizes the settings with various numbers of incoming tasks. Due to the page limits, we only present representative experimental results obtained from the settings, where the number of the computation unit categories is four.

Our experiments compare EFRO against alternative strategies, depending on specific evaluation goals. In Setting 1, for example, we examine the performance discrepancies between EFRO and the other three approaches, namely, the DECM [16], MESF [20], and Round-Robin (RR) strategies [21].

Setting 2 has a handful of sub-settings with distinct configurations in both the number of tasks and the category types of computation units. Table 1 also outlines a group of settings for the second group of experiments. The main purpose of this set of experiments is to study the impact of available computing resources when the number of tasks introduced is static in a fog computing environment. The competitors under this setting are different from those tested under Setting 1. We compare EFRO with MESF and RR, because the dynamic-programming-based DECM is inapplicable to situations where more computation units types become available. In this case, running a dynamic programming takes a dramatically long time to produce optimal allocation decisions, making the performance comparisons irrelevant. The next section presented some experiment results base on our collections and analyses.

5.2 Experiment Results

Now we are positioned to discuss the main findings and observations deriving from the experiments.

We introduce a performance metric to quantify the performance of EFRO improved over the existing techniques. The metric is a ration between the energy cost of an alternative strategy and that of our EFRO. Such a ratio is defined as parameter *R*. More specifically speaking, we have *R1=DECM/EFRO*, *R2=MESF/EFRO*, and *R3=RR/EFRO*. The meaning of *R* is translated into the following statements: EFRO has an equal energy cost to the comparing approach when $R=1$; EFRO has less energy cost to the competitive approach when $R>1$; EFRO has more energy cost than the alternative approach when $R<1$. Figs. 4, 5, and 6 plot experiment results collected under Settings 1-1, 1-2, and 1-5. Data are sorted by the *R2* values to intuitively illustrate the performance trends. We observe that our EFRO mostly reach the optimal solution

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

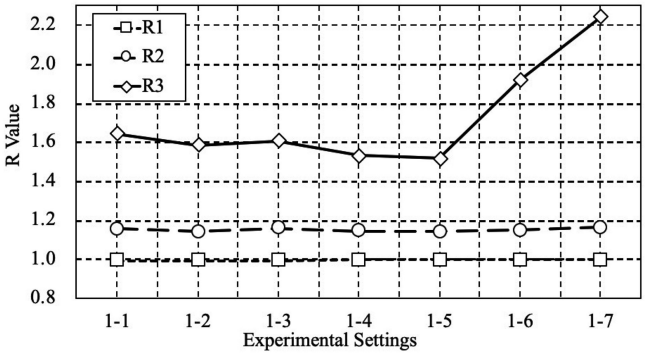


Fig. 7. Average R values under Setting 1.

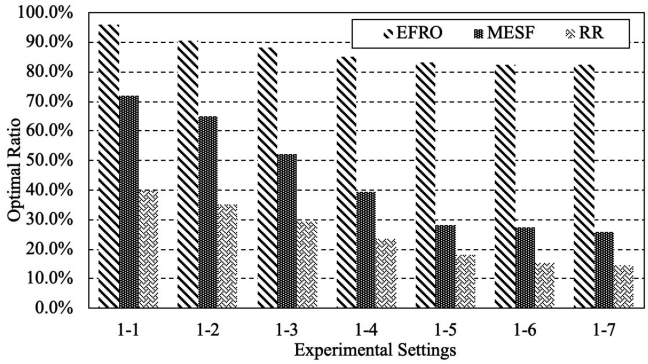


Fig. 9. Ratio of obtaining optimal solutions for EFRO under Setting 1.

performance, delivering a visible advantage compared against the other non-optimal solutions.

Next, we show the average R values in Fig. 7, which indicates that the energy-efficiency benefits gained by our EFRO becomes more pronounced. Statistically speaking, the scope of average *R1* is [0.9941, 0.9970], suggesting marginal energy-saving differences between EFRO and DECM. The scopes of the average *R2* and *R3* values are [1.1404, 1.1633] and [1.5155, 2.2441]; The main takeaway from these results is that an optimal or a near-optimal solution is necessary due to the noteworthy superiority. A sub-optimal solution could not have a similar performance to an optimal/near-optimal solution, even though the greedy algorithm offers improvements to a certain extent. Fig. 8 depicts comparisons in terms of average energy cost under Setting 1. From Fig. 8 we could observe that with the increasing number of tasks, the energy cost continued to increase. The energy cost of EFRO and DECM which had remained close was the least, the energy cost of MESF was the second, and the energy cost of RR was the most. In order to show that our proposed algorithm can obtain an approximate optimal solution, we compared the three strategies EFRO, MESF and RR. We compared the approximate solution obtained by the EFRO, MESF and RR with the optimal solution obtained by the DECM, and calculated the same ratio between them to obtain Fig. 9. The ratios of obtaining optimal solutions by EFRO under Settings 1-1 to 1-7 are 96.0, 90.6, 88.1, 85.0, 83.1, 82.6 and 82.3 percent, respectively. As can be seen from Fig. 9, although the approximate ratio is decreasing with the increase in the number of tasks, our proposed algorithm was still a bit higher than the other two. Therefore, we have reason to believe that our algorithm is near-optimal.

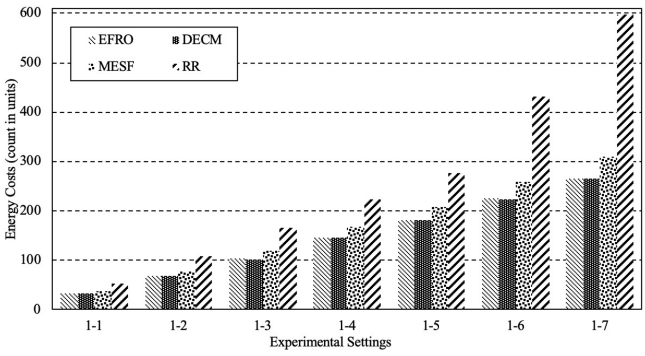


Fig. 8. Comparisons of average energy costs under Setting 1. Energy costs are gauged in units.

We also noticed that the optimal solution rates slightly went down when the size of the inputs were increasing. An analysis was given here to explain the phenomenon. Since our approach is a hill-climbing mechanism, which tried to transfer the non-optimal to the optimal strategy. A few adjustments were made during the process of the transfer, as called *Smart Shifting*. For each adjustment, our scheme always primarily selected the exchanging allocation pairs with the biggest improvement in order to shorten the time cost. This mechanism resulted in having chances to encounter the local optimal trap. The smart shifting operation would be ended when no more better exchanging option was found. However, since some exchanging operations with less improvements were bypassed, some "climbing" options would be missed and the transfer might be terminated before reaching the optimal solution. With the growing complexity of the input, the probability of local optimal solutions would be increased. Our experiment results also demonstrated the analysis.

We divide the optimal solutions by the energy difference between EFRO and DECM to further highlight our technique's performance edge in energy conservation. The mathematical expression of this metric is $(EFRO - DECM)/DECM$. This metric assesses whether the solution constructed by the proposed model is approximates to an optimal solution. A low ratio value means a solution is close to an optimal solution. Fig. 10 displays the average ratio values tested under the seven sub-settings. We observe that the average ratio values are remarkably low; the average ratios are in a range between 2.97‰ and 5.86‰ .

We notice that adopting dynamic programming to construct optimal solutions generally takes a long time period; thus, assessing the time length of the allocation plan generation is a crucial part in our experiments. Figs. 11 and 12 plot experiment results of the algorithms tested under settings 1-2, and 1-5. The data are sorted by with respective o MESF's time in an ascending order to improve the presentation of the results. The figures depict that dynamic programming requires apparently long time to construct allocation plans; the time length are remarkably impacted by the number of incoming tasks. We analyze representative statistics to show the generation-time differences. For example, under Setting 1-1, DECM's average plan-generation time is 2.19 times longer than that of EFRO; and EFRO's is 4.26 times and 5.32 times longer than MESF's and RR's generation times, respectively. Compared with DECM under Setting 1-5, our EFRO

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

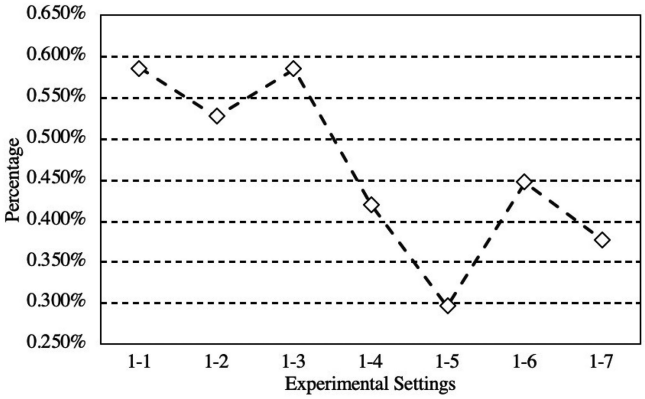


Fig. 10. The energy-saving performance of EFRO under Setting 1.

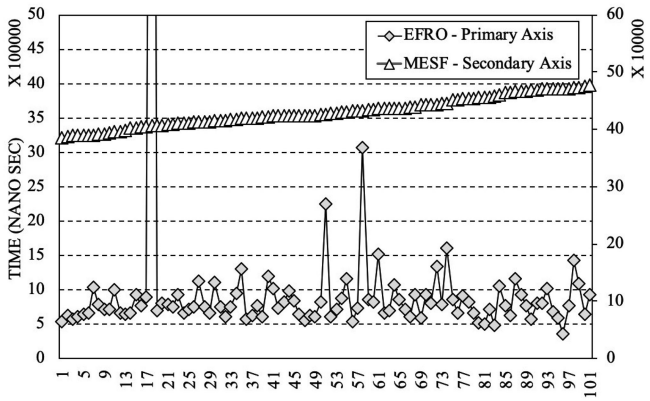


Fig. 11. Comparisons of allocation-plan generation time between EFRO and MESF under Setting 1-2.

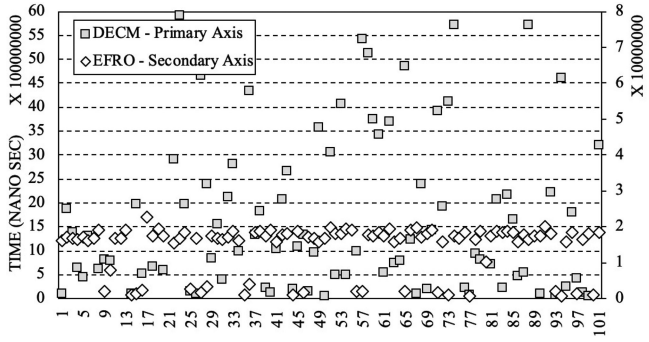


Fig. 12. Comparison of allocation-plan generation time between EFRO and DECM under Settings 1-5.

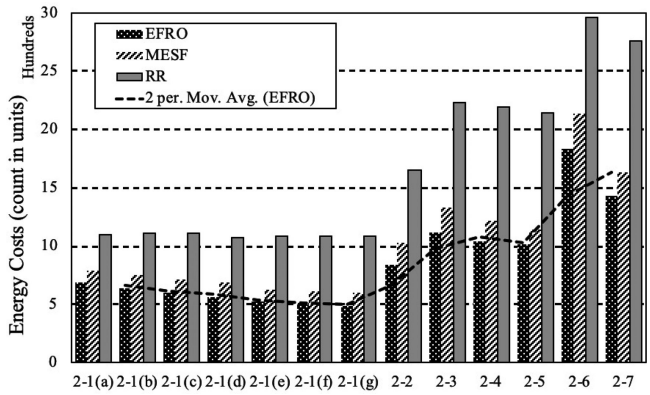


Fig. 13. Average energy-cost comparisons under Setting 2.

speedups the allocation-plan construction performance by a factor of 506.59 on average. Compared with dynamic programming, our approach doesn't grow in a high rate; EFRO's generation time is 31.62 and 44.15 times than those of MESF and RR, respectively.

Now we evaluate workload scenarios where there are a massive number of incoming tasks and various numbers of computation units. Fig. 13 unveil performance results of the tested strategies under Setting 2. Fig. 13 depicts the comparison results of average energy costs, which suggest that our approach saves a great deal of energy under distinct workload conditions. For instance, under Setting 2-1(a), our EFRO conserves energy by 12.42 and 36.94 percent compared with MESF and RR, respectively. Under Setting 2-2, the energy-saving rates of EFRO become 18.28 and 49.08 percent compared with MESF and RR, respectively. Results obtained from the same comparison applies to Setting 2-7 are 12.62 and 48.31 percent. To sum it up, we demonstrate that EFRO achieves a superb energy-saving performance under a wide range of workload conditions.

In a nutshell, our experimental results show an array of intriguing findings. First, EFRO is adroit at delivering high energy-saving performance. Compared against the other non-optimal solutions, EFRO's advantage with respective to energy conservation become appealing to fog/edge computing environments, because these systems generally have a complicated networking setups for service providers and clients. We compared our proposed algorithm with a dynamic programming algorithm in terms of energy and time consumption under 10×10^3 different experimental settings. The experimental results unravel that our algorithm is close to the optimal solution. Second, our proposed EFRO offers an acceptable allocation-plan generation time, which is superior to that of the dynamic programming scheme. The experimental results show evidence that EFRO is competent to efficiently deal with complicated workload scenarios. Finally, both the numbers of tasks and the computation units impose significant impacts on the plan-generation efficiency. It is of necessity that fog servers should governs a pre-stored table or manages a dynamic table to cope with various workload conditions. We will dig deep into the implementation of a dynamic table in a future project.

6 Conclusion

In this study, we developed an energy-efficient resource scheduling technique - *EFRO* - for fog computing systems running applications with timing constraints. The system model of EFRO embraces three components, namely, the cloud, fog, and edge layers. EFRO seamlessly integrates the standardization and smart shift operations coupled with a hill-climbing mechanism to construct near-optimal resource allocation solutions, in which both energy consumption and scheduling length are minimized with a guaranteed time constraint.

Our conducted extensive experiments to demonstrate evidence that our EFRO approach is adept at making efficient and judicious task-allocation decisions at fog servers. EFRO achieves superb energy-saving and plan-generation performance under a wide range of workload conditions.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

More specifically, our EFRO improves the energy efficiency of the existing *MESF* and *RR* schemes by 54.83 and 71.28 percent, respectively. EFRO cuts back DECM's time spent in generating allocation plans by up to a factor of 507.

In our future work, we are enthusiastic about extending this study from three dimensions. First, we will pilot a way of implementing multiple resource allocators rather than a single- allocator mechanism. Augmenting one additional allocator or more will dramatically increase the difficulty level of the resource allocation problem. The key issue is to develop a multi-allocator-based resource manager that assigns computing resources to tasks in heterogeneous Fog system at a low cost. Second, we have a solid plan to seamlessly integrate the reinforcement learning technique with our strategy in an effort to cope with a implementation scenario that is continuously changing. We reckon that a scheme powered by caches will be probed to strengthen the adoptability of our proposed resource allocation strategy. Finally, we expect to dig deep into a framework that introduces an option to on-ground examinations in real-world applications.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation under Grant IIS-1618669, and Grant OAC-1642133, CCF-0845257 (CAREER) (Dr. Xiao Qin), in part by National Natural Science Foundation of China under Grant 61972034, in part by Natural Science Foundation of Beijing Municipality under Grant 20D20116, in part by Natural Science Foundation of Shandong Province under Grant ZR2019ZD10, in part by Guangxi Key Laboratory of Cryptography and Information Security under Grant GCIS201803, and in part by Henan Key Laboratory of Network Cryptography Technology under Grant LNCT2019-A08. Dr. Keke Gai was supported by Beijing Institute of Technology Research Fund Program for Young Scholars.

References

[1] L. Wang, J. Zhan, W. Shi, and Y. Liang, "In cloud, can scientific communities benefit from the economies of scale?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 2, pp. 296–303, Feb. 2012.

[2] F. Tao, Y. Zuo, X. Da, and L. Zhang, "IoT-based intelligent perception and access of manufacturing resource toward cloud manufacturing," *IEEE Trans. Ind. Informat.*, vol. 10, no. 2, pp. 1547–1557, May 2014.

[3] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the Internet of Things," *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 144–149, Dec. 2012.

[4] X. Xu and Q. Hua, "Industrial big data analysis in smart factory: Current status and research strategies," *IEEE Access*, vol. 5, pp. 17543–17551, 2017.

[5] L. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[6] C. Chen and C. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Inf. Sci.*, vol. 275, pp. 314–347, 2014.

[7] L. Wang *et al.*,"BigDataBench: A big data benchmark suite from internet services," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Architecture*, 2014, pp. 488–499.

[8] G. Yan, D. Wen, S. Olariu, and M. Weigle, "Security challenges in vehicular cloud computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 1, pp. 284–294, Mar. 2013.

[9] B. Soelistijanto and M. Howarth, "Transfer reliability and congestion control strategies in opportunistic networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 538–555, First Quarter 2014.

[10] K. Gai, M. Qiu, and X. Sun, "A survey on FinTech," *J. Netw. Comput. Appl.*, vol. 103, pp. 262–273, 2018.

[11] H. Ngo, E. Larsson, and T. Marzetta, "Energy and spectral efficiency of very large multiuser MIMO systems," *IEEE Trans. Commun.*, vol. 61, no. 4, pp. 1436–1449, Apr. 2013.

[12] C. Perera, C. Liu, and S. Jayawardena, "The emerging Internet of Things marketplace from an industrial perspective: A survey," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 4, pp. 585–598, Dec. 2015.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[14] J. Cao, L. Xu, R. Abdallah, and W. Shi, "EdgeOS_H: A home operating system for Internet of Everything," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1756–1764.

[15] Q. Zhang, X. Zhang, and W. Shi, "Firework: Big data processing in collaborative edge environment," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2016, pp. 81–82.

[16] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *J. Netw. Comput. Appl.*, vol. 59, pp. 46–54, 2015.

[17] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency Comput. Pract. Experience*, vol. 29, no. 16, 2017, Art. no. e3975.

[18] Y. Jiao, P. Wang, D. Niyato, and K. Suankaewmanee, "Auction mechanisms in cloud/fog computing resource allocation for public blockchain networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 1975–1989, Sep. 2019.

[19] M. Chiang, S. Ha, I. Chih-Lin , F. Risso, and T. Zhang, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 18–20, Apr. 2017.

[20] Z. Dong, N. Liu, and R. Rojas-Cessa , "Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers," *J. Cloud Comput.*, vol. 4, no. 1, 2015, Art. no. 5.

[21] E. Hahne,"Round-robin scheduling for max-min fairness in data networks," *IEEE J. Sel. Areas Commun.*, vol. 9, no. 7, pp. 1024–1039, 1991.

[22] Q. He, C. Zhang, X. Ma, and J. Liu, "Fog-based transcoding for crowdsourced video livecast," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 28–33, Apr. 2017.

[23] C. Wang, C. Liang, F. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.

[24] H. Zhang, Y. Zhang, Y. Gu, D. Niyato, and Z. Han, "A hierarchical game framework for resource management in fog computing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 52–57, Aug. 2017.

[25] K. Dantu, S. Ko, and L. Ziarek, "RAINA: Reliability and adaptability in android for fog computing," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 41–45, Apr. 2017.

[26] R. Deng, R. Lu, C. Lai, T. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[27] S. Li, M. Maddah-Ali, and A. Avestimehr, "Coding for distributed fog computing," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 34–40, Apr. 2017.

[28] L. Shen, Z. Zhang, and P. An, "Fast CU size decision and mode decision algorithm for HEVC intra coding," *IEEE Trans. Consum. Electron.*, vol. 59, no. 1, pp. 207–213, Feb. 2013.

[29] X. Masip-Bruin, E. Mar ín-Tordera, G. Tashakor, A. Jukan, and G. Ren, "Foggy clouds and cloudy fogs: A real need for coordinated management of fog-to-cloud computing systems," *IEEE Wireless Commun.*, vol. 23, no. 5, pp. 120–128, Oct. 2016.

[30] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 1, pp. 108–119, Jan.–Mar. 2017.

[31] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2018.2890603.

[32] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.

[33] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.
[34] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.
[35] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.
[36] K. Gai, M. Qiu, M. Chen, and H. Zhao, "SA-EAST: Security-aware efficient data transmission for ITS in mobile heterogeneous cloud computing," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, 2017, Art. no. 60.
[37] S. Bitam, A. Mellouk, and S. Zeadally, "VANET-cloud: A generic cloud computing model for vehicular Ad Hoc networks," *IEEE Wireless Commun.*, vol. 22, no. 1, pp. 96–102, Feb. 2015.
[38] X. Chen and L. Wang, "Exploring fog computing-based adaptive vehicular data scheduling policies through a compositional formal method? PEPA," *IEEE Commun. Lett.*, vol. 21, no. 4, pp. 745–748, Apr. 2017.
[39] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez, "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3113–3125, Apr. 2019.
[40] M. Moltafet, S. Parsaeefard, M. Javan, and N. Mokari, "Robust radio resource allocation in MISO-SCMA assisted C-RAN in 5G networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 5758–5768, Jun. 2019.
[41] J. Shi, W. Yu, Q. Ni, W. Liang, Z. Li, and P. Xiao, "Energy efficient resource allocation in hybrid non-orthogonal multiple access systems," *IEEE Trans. Commun.*, vol. 67, no. 5, pp. 3496–3511, May 2019.
[42] M. Mohseni, S. Banani, A. Eckford, and R. Adve, "Scheduling for VoLTE: Resource allocation optimization and low-complexity algorithms," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1534–1547, Mar. 2019.
[43] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing-based face identification and resolution scheme in Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1910–1920, Aug. 2017.
[44] J. Chen, L. Zhang, Y. Liang, X. Kang, and R. Zhang, "Resource allocation for wireless-powered IoT networks with short packet communication," *IEEE Trans. Wireless Commun.*, vol. 18, no. 2, pp. 1447–1461, Feb. 2019.
[45] A. Soltanian, D. Naboulsi, R. Glitho, and H. Elbiaze, "Resource allocation mechanism for media handling services in cloud multimedia conferencing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1167–1181, May 2019.
[46] Y. Huang, "Computing quantum discord is NP-complete," *New J. Phys.*, vol. 16, no. 3, 2014, Art. no. 033027.
[47] H. Sun, P. Stolf, and J. Pierson, "Spatio-temporal thermal-aware scheduling for homogeneous high-performance computing datacenters," *Future Gener. Comput. Syst.*, vol. 71, pp. 157–170, 2017.
[48] M. Chen, Y. Hao, Y. Li, C. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: Architecture and service modes," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 18–24, Jun. 2015.
[49] C. Wu, R. Chang, and H. Chan, "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters," *Future Gener. Comput. Syst.*, vol. 37, pp. 141–147, 2014.
[50] Y. Guo, Y. Gong, Y. Fang, P. Khargonekar, and X. Geng, "Energy and network aware workload management for sustainable data centers with thermal storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2030–2042, Aug. 2014.
[51] K. Gai, M. Qiu, and H. Zhao, "Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing," *J. Parallel Distrib. Comput.*, vol. 111, pp. 126–135, 2018.
[52] K. Gai, M. Qiu, and H. Zhao, "Privacy-preserving data encryption strategy for big data in mobile cloud computing," *IEEE Trans. Big Data*, to be published, doi: 10.1109/TBDATA.2017.2705807.
[53] K. Gai, M. Qiu, and H. Zhao, "Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2016.2594172.



Keke Gai (Senior Member, IEEE) received the BEng degree majored in automation, from the Nanjing University of Science and Technology, Nanjing, China, in 2004, the MET degree in educational technology from the University of British Columbia, Vancouver, British Columbia, Canada, in 2010, the MBA degree in business administration, in 2009, the MS degree in information technology from Lawrence Technological University, Southfield, Michigan, in 2014, and the PhD degree in computer science from Pace University, New York. He is currently an associate professor with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. He has published three books and more than 110 peer-reviewed journal/conference papers. He has been granted five IEEE best paper awards (e.g., TrustCom 18' and HPCC 18') and two IEEE best student paper awards in recent five years. His research interests include cyber security, edge computing, cloud computing, blockchain, and reinforcement learning.



Xiao Qin (Senior Member, IEEE) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, China, and the PhD degree in computer science from the University of Nebraska-Lincoln, Lincoln, in 1992, 1999, and 2004, respectively. Currently, he is a professor with the Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama. His research interests include parallel and distributed systems, energy-efficient computing, storage systems, fault tolerance, real-time systems, and performance evaluation. He received the U.S. NSF CAREER Award, in 2009.



Liehuang Zhu (Member, IEEE) received the BE and ME degrees from Wuhan University, Wuhan, China, in 1998 and 2001, respectively, and the PhD degree in computer science from the Beijing Institute of Technology, Beijing, China, in 2004. He is currently a professor with the School of Computer Science & Technology, Beijing Institute of Technology, Beijing, China. He has published more than 100 peer-reviewed journal or conference papers, including more than ten IEEE/ACM Transactions papers (*IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Vehicular Technology*, *IEEE Transactions on Smart Grid*, *Information Sciences*, *IEEE Network*, *Computer & Security*, etc.). He has been granted a number of IEEE best paper awards, including IWQoS 17', TrustCom 18'. His research interests include security protocol analysis and design, wireless sensor networks, and cloud computing.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:37:34 UTC from IEEE Xplore. Restrictions apply.