A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing

Yongxuan Sang [1] , **Junqiang Cheng** [2] , **Bo wang** [Corresp., 1] , **Ming Chen** [1]

[1] Zhengzhou University of Light Industry, Zhengzhou, China

[2] Europe-Aisa Hi-tech and Digital Technology Company Limited, Zhengzhou, China

Corresponding Author: Bo wang
Email address: wangb@zzuli.edu.cn

In this paper, we study on the task scheduling problem to optimize the Service Level Agreement (SLA) satisfaction in device-edge-cloud cooperative computing. We first formulate the problem into a binary nonlinear programming, and then propose a heuristic scheduling method with three stages to solve the problem in a reasonable time. The first stage of the proposed method is trying to fully exploit the abundant cloud resources, by scheduling user tasks in the resource priority order of clouds, edge servers, and local devices, to satisfy as many SLA requirements as possible. In the second stage, the proposed heuristic method reschedules some tasks from edges to devices with SLA satisfactions, to provide more available shared edge resources for other tasks cannot be completed locally, and schedules these tasks to edge servers. At the last stage, our method reschedules as many tasks as possible from clouds to edges or devices, to improve the resource cost. Experiment results show that our method has up to 59% better performance in SLA satisfaction without decreasing the resource efficiency, compared with eight of classical methods and state-of-the-art methods.

A Three-Stage Heuristic Task Scheduling for Optimizing the Service Level Agreement Satisfaction in Device-Edge-Cloud Cooperative Computing

Yongxuan Sang[1], Junqiang Cheng[2], Bo Wang[1], and Ming Chen[1]

[1]**Software Engineering College, Zhengzhou University of Light Industry, No.136 Kexue Road, Zhengzhou, China, 450000**

[2]**Europe-Aisa Hi-tech and Digital Technology Company Limited, Zhengzhou, China, 450000**

Corresponding author:

Bo Wang[3]

Email address: wangb@zzuli.edu.cn

ABSTRACT

Device-edge-cloud cooperative computing is increasingly popular as it can effectively address the problem of the resource scarcity of user devices. It is one of the most challenging issues to improve the resource efficiency by task scheduling in such computing environments. Existing works used limited resources of devices and edge servers in preference, which can lead to not full use of the abundance of cloud resources. This paper studies on the task scheduling problem to optimize the service level agreement satisfaction in terms of the number of tasks whose hard-deadlines are met for device-edge-cloud cooperative computing. This paper first formulates the problem into a binary nonlinear programming, and then proposes a heuristic scheduling method with three stages to solve the problem in polynomial time. The first stage is trying to fully exploit the abundant cloud resources, by pre-scheduling user tasks in the resource priority order of clouds, edge servers, and local devices. In the second stage, the proposed heuristic method reschedules some tasks from edges to devices, to provide more available shared edge resources for other tasks cannot be completed locally, and schedules these tasks to edge servers. At the last stage, our method reschedules as many tasks as possible from clouds to edges or devices, to improve the resource cost. Experiment results show that our method has up to 59% better performance in service level agreement satisfaction without decreasing the resource efficiency, compared with eight of classical methods and state-of-the-art methods.

INTRODUCTION

With the development of computer and communications technology as well as the growing need for the quality of human life, smart devices, e.g., smartphones and Internet of Thing (IoT) devices, have become more and more popular. As shown in Cisco Annual Internet Report (CAIR) (Cisco, 2020) released in March 2020, networked devices will be increased from 18.4 billion in 2018 to 29.3 billion in 2023, and IoT devices will account for 50 percent by 2023. Juniper Research has reported similar results in 2018, IoT devices will grow at 140% over the next 4 years (Sorrel, 2018). Because of limited resources, plenty of user devices cannot satisfy their respective requirements at most time (Ghasempour, 2019; Liu et al., 2019b), as Internet applications has undergone rapid growth in both variety and complexity with the development of artificial intelligence algorithms (e.g., deep neural networks) and communication technologies (e.g., 5G and Wifi 6) (Wang et al., 2020b).

To address the resource scarcity problem of user devices, several researchers exploited the low latency edge resources (Balasubramanian et al., 2020) and the abundant cloud resources (Strumberger et al., 2019). While, only one of them cannot address the problem effectively due to either the limited resource of edges or the poor network performance of clouds (Wang et al., 2019b). Thus, with the integration

45 of respective benefits of edge and cloud computing, device-edge-cloud cooperative computing (DE3C)
46 (Hong et al., 2019) is an effective way, where edges and clouds are employed jointly for expanding the
47 resource capacity of user devices.
48   Task scheduling or offloading is an effective way for optimizing the task performance and the resource
49 efficiency for DE3C, which decides the location (the corresponding device, an edge or a cloud) where each
50 task to be processed (offloading decision) and the computing resources which each task performs on in a
51 specified order (task assignment and ordering) (Wang et al., 2020b; Islam et al., 2021). Therefore, several
52 works have proposed various task scheduling methods trying to optimize the response time (Han et al.,
53 2019; Meng et al., 2019, 2020; Apat et al., 2019; Ren et al., 2019; Liu et al., 2019a; Wang et al., 2021),
54 the resource cost (Mahmud et al., 2020; Gao et al., 2019; Chen et al., 2019) or the profit (Chen et al.,
55 2020; Yuan and Zhou, 2020) for providing services in DE3C. While, these works concerned on addressing
56 only one or two sub-problems of task scheduling, e.g., offloading decision or/and task assignment, and
57 thus cannot provide global optimal solutions. In addition, a lot of existed works did not employ local
58 device resources without network latency, even though a lot of smart devices have been equipped with
59 computing resources almost equivalent to personal computers (Wu et al., 2019), nowadays.

60 **Motivation**
61 This paper focuses on the joint problem of offloading decision, task assignment and task ordering, to
62 optimize the profit for service providers in DE3C, by improving the Service Level Agreement (SLA)
63 satisfaction. An SLA is enforced when a user uses a provider's service. If the SLA is fail to be satisfied, the
64 provider must pay a penalty, which reduces the provider's profit. In addition, an SLA violation reduces the
65 provider's reputation, and thus may lead to a loss of some potential users (Papadakis-Vlachopapadopoulos
66 et al., 2019). While plenty of related works only concerned on improving the response time or the resource
67 cost, which is in contradiction to the SLA satisfaction optimization, as better response times or less
68 resources generally result in a less number of completed tasks. In addition, to our best knowledge, all
69 existed work prioritised local resources (devices) for processing tasks and rented resources from clouds
70 only when local and edge resources are not enough, as local resources are cheap and have no network
71 latency. They didn't consider that some tasks processed locally or in edges can be assigned to clouds to
72 save some local or edge resources for completing tasks that cannot be finished by cloud resources.
73   For example, there are four tasks, t_1, t_2, t_3, and t_4, to be scheduled in a DE3C with one device, one edge
74 server, and a cloud. The information of tasks and resources are shown in Table 1. The times consumed by
75 t_1 and t_2 are 50s, 51s, and 35s, respectively, when they are scheduled to the device, the edge server, and the
76 cloud, and the consumed times are 5s, 6s, and 12.5s, respectively, for t_3 and t_4. With the scheduling order
77 of t_1, t_2, t_3, and t_4, and the idea of using device resources first, t_1, t_3, and t_2 are respectively scheduled to
78 the device, the edge server, and the cloud, but the requirements of t_4 cannot be satisfied. But if scheduling
79 tasks with the priority order of the cloud, the edge server, and the device, requirements of all tasks can be
80 satisfied, where t_1 and t_2 are scheduled to the cloud and t_3 and t_4 are scheduled to the device.

Table 1. The information of DE3C system for the case motivating our work.

a. task requirements			
Task	Computing resource amount	Transferred data amount	Deadline
t_1	100GHz	100Mbit	50s
t_2	100GHz	100Mbit	50s
t_3	10GHz	100Mbit	10s
t_4	10GHz	100Mbit	10s

b. resource configurations		
Resource	Computing capacity	Transmission bandwidth
Device	2GHz	∞
Edge	2GHz	100Mbps
Cloud	4GHz	10Mbps

81 **Contribution**
82 This paper focuses on maximizing the SLA satisfaction, i.e., optimizing the number of tasks whose
83 hard-deadlines are met, in DE3C by task scheduling, by exploiting both the low network latency of

edges and the rich computing resources of clouds. To address the task scheduling problem in DE3C, the paper formulates it into a binary nonlinear programming (BNLP) for the SLA satisfaction optimization. And to solve the problem in polynomial time, a heuristic method is designed based on the idea of least accumulated slack time first (LASTF) (Wang et al., 2020a) and earliest deadline first (EDF) (Benoit et al., 2021). In brief, the contributions of this paper can be summarized as followings.

- The task scheduling problem of DE3C is formulated as a BNLP with two objectives, where the major one is maximizing the number of tasks whose requirements are satisfied[1] and the second one is maximizing the resource utilization.

- A three-stage heuristic task scheduling method (TSSLA) is designed for DE3C. The first stage is to exploit the abundant computing resource of clouds, to finish as many tasks as possible[2], by pre-scheduling tasks in the resource priority order of clouds, edges, and devices. In the second stage, TSSLA reschedules tasks from edges to respective devices to make some edge resources free for completing more tasks, and schedules remaining unscheduled tasks to edges. At the last stage, our method reschedules tasks from clouds to respective devices or edges, to reduce the resource cost. For task scheduling in each location, the proposed method respectively employs LASTF and EDF to assign the computing core for each task and decide the task execution order.

- Simulated experiments are conducted referring to recent related works and the reality, to evaluate our proposed heuristic method. Experiment results verify our method having a much better performance than eight of classical and state-of-the-art methods in optimizing SLA satisfaction.

The rest of this paper is organized as follows. The second section illustrates the related works. Third section present the formulation of the task scheduling problem this paper concerned. Fourth section presents the three-stage heuristic method designed, and analyses its time complexity. The subsequent section evaluates the performance of our task scheduling method by simulated experiments. And finally, the last section concludes this paper.

RELATED WORK

As DE3C is one of the most effective ways to solve the problem of insufficient resources of smart devices and task scheduling is a promising technology to improve the resource efficiency, several researchers have focused on the design of efficient task scheduling methods in various DE3C environments (Wang et al., 2020b).

To improve the response time of tasks, the method proposed by (Apat et al., 2019) iteratively assigned the task with the least slack time to the edge server closest to the user. Tasks are assigned to the cloud when they cannot be finished by edges. Their work didn't consider the task scheduling on each server. OnDisc, proposed by (Han et al., 2019), heuristically dispatched a task to the server providing the shortest additional total weighted response time (WRT), and sees the cloud as a server, to improve overall WRT. (Stavrinides and Karatza, 2019) proposed a heuristic method for improving the deadline miss Ratio. Their proposed method respectively employed EDF and earliest finish time first for task selection and resource allocation, and tried to fill a task before the input data is ready for the next task to execute.

Above research focused on the performance optimization for task execution, while didn't concern the cost of used resources. In general, a task requires more resources for a better performance, and thus there is a trade-off between the task performance and the resource cost. Therefore, several works concerned the optimization of the resource cost or the profit for service providers. For example, (Chen et al., 2020) presented a task scheduling method to optimize the profit, where the value of a task was proportional to the resource amounts and the time it took, and resources were provided in the form of VM. Their proposed method first classified tasks based on the amount of its required resources by K-means. Then, their method used Kuhn-Munkres method to solve the optimal matching of tasks and VMs with profit maximization for the VM class and the task class closest to the VM class, where all VMs were seen as one VM class. This work ignored the heterogeneity between edge and cloud resources, which may lead to resource inefficiency (Kumar et al., 2019). (Li et al., 2020) tried to optimize the finish time and the cloud

---

[1]This paper considers the number of finished tasks with hard-deadlines as the SLA satisfaction metric. The proposed approach is compatible to any other metric.

[2]In this paper, finishing/completing a task means finishing the task within its deadline, i.e., satisfying the task's SLA requirement.

resource usage cost. Their proposed method first made the offloading decision for each task, adopting the artificial fish swarm algorithm improved by simulated annealing method for calculating the probability of updating bulletin, to avoid falling into local optimal solution. Then their method greedily assigned a task to the computing node with the minimum utilization in edges or the cloud. This work focused on media delivery applications, and thus considered that each task can be divided into multiple same-sized subtasks for parallel process, which limited its application scope. The method proposed by (Mahmud et al., 2020) iteratively assigned the offloaded application to the first computational instance such that all requirements are satisfied and the profit merit is minimum, where cloud-based instances were sorted behind edge-based instances and the profit merit was defined as the ratio of the profit and the slack time.

All of the aforementioned methods employed only edge and cloud resources for task processing, even though most of user devices have been equipped with various computing resources (Wu et al., 2019) which have zero transmission latency for users' data. To exploit all the advantages of the local, edge and cloud resources, some works are proposed to address the task scheduling problem for DE3C. The method presented in (Lakhan and Li, 2019) first tried several existed task order method, e.g., EDF, EFTF, and LSTF, and selected the result with the best performance for task order. Then, the method used existed pair-wise decision methods, TOPSIS (Liang and Xu, 2017) and AHP (Saaty, 2008), to decide the position for each task's execution, and applied a local search method exploiting random searching for the edge/cloud. For improving the delay, the approach presented in (Miao et al., 2020) first decided the amounts of data that is to be processed by the device and an edge/cloud computing node, assuming each task can be divided into two subtasks with any data size. Then they considered to migrate some subtasks between computing nodes to further improve the delay, for each task. The method proposed in (Zhang et al., 2019) iteratively assigned the task required minimal resources to the nearest edge server that can satisfy all of its requirements. Ma et al. Ma et al. (2022) proposed a load balance method for improving the revenue for edge computing. The proposed method allocated the computing resources of the edge node with the most available cores and the smallest move-up energy to the new arrived task. To improve the total energy consumption for executing deep neural networks in DE3C with deadline constraints, Chen et al. Chen et al. (2022) proposed a particle swarm optimization algorithm using mutation and crossover operators for population update. Wang et al. Wang et al. (2021) leveraged reinforcement learning with sequence-to sequence neural network for improving the latency and the device energy in DE3C. Machine learning-based or metaheuristic-based approaches may achieve a better performance than heuristics, but in general, they consume hundreds to tens of thousands more time, which makes them not applicable to make online scheduling decisions.

All of these existed research concerned only one or two problems of offloading decision, task assignment, and task ordering, which leads to suboptimal solutions. In addition, they were not fully explored the advantage of abundant cloud resources, as they considered assigning tasks to the cloud only when local and edge resources are exhausted. To address these issues, this paper designs a heuristic method for optimizing SLA satisfaction in DE3C. To our best knowledge, this is the first attempt to jointly address the problems of offloading decision, task assignment, and task ordering.

PROBLEM FORMULATION

The DE3C environment considered in this paper is composed of various user devices, multiple edges (short for edge computing centers), and one cloud [3], as shown in Fig 1. Each device launches one or more tasks for processing data it collected from user behaviors or surroundings. Each task can be processed locally, or offloaded to an edge covering the device or the cloud. When a task is offloaded, its processed data must be transmitted from the device to the edge or the cloud before its processing, by various communication links, e.g., wireless networks, telecommunications, etc. There have been some research on data transmission in advance to improve the network latency by predicting task offloading decisions (Zhang et al., 2017), which is complementary to our work. In this paper, the data is assumed to be transmitted only after the offloading decision is made for each task, which can avoid the waste of network resources due to the failure of predicting.

The considered DE3C system is consisted of M user devices ($\mathscr{M} = \{m_1, m_2, .., m_M\}$), E edges ($\mathscr{E} = \{e_1, e_2, ..., e_E\}$), and one public cloud. For device m_j, there are n_j cores ($\mathscr{CM}_j = \{cm_{j,1}, cm_{j,2}, ..., cm_{j,n_j}\}$), and each core has g_j computing capacity. In edge e_k, s_k servers are deployed, which are represented as

---

[3]Multiple clouds can be seen as one big public cloud including the resources provisioned by these clouds.

ES: Edge Server(s)   CS: Cloud Servers   AP: Access Point   MaBS: Macro Base Station
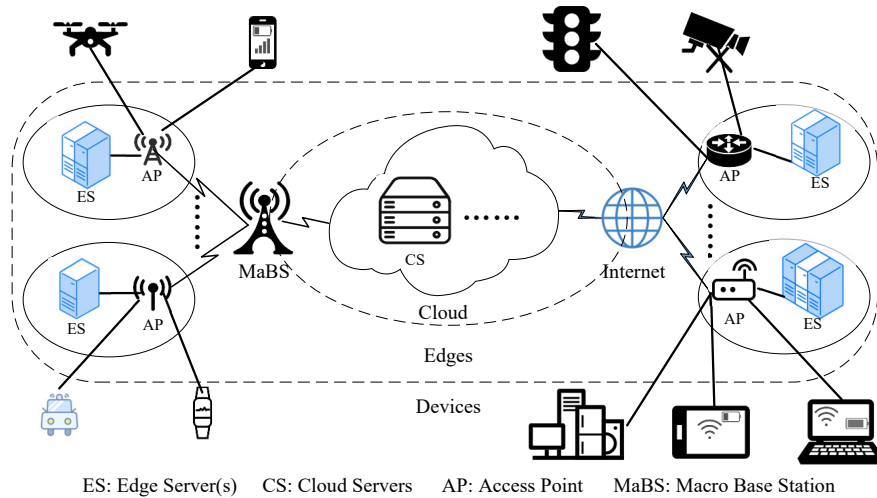
Figure 1. The DE3C environment considered in this paper.

$\mathscr{S}_k = \{s_{k,1}, s_{k,2}, ..., s_{k,s_k}\}$. Each edge server (say $s_{k,l}$) has $n_{k,l}$ cores, each with $g_{k,l}$ computing capacity. For satisfying users' requirements when local and edge resources are not enough, assuming there are V cloud servers[4] ($\mathscr{V} = \{v_1, v_2, ..., v_V\}$) rented from the cloud. Cloud server v_r has n_{v_r} cores, and the capacity of each core is g_{v_r}. The price of v_r is p_r per unit time. The bandwidth of transmitting data from device m_j to $s_{k,l}$ and v_r are represented as $b_{j,k,l}$ and $b_{j,r}$, respectively, which can be easily calculated by transmission channel state data (Chen et al., 2016; You et al., 2017; Du et al., 2019). If a device is not in the coverage of an edge, corresponding bandwidths are set to infinity.

Assuming there are T tasks, $\mathscr{T} = \{t_1, t_2, ..., t_T\}$, requested by users for processing in the DE3C, and each task can be processed in the device launching it (locally), an edge server communicated with the device, or a cloud server. For each task, say t_i, it has in_i data must be processed, and has f_i processing length. To make our method wider application, assuming the data size to be processed and the processing length are independent for each task in any computing node. Task t_i must be finished before its deadline d_i defined by corresponding SLA[5]. Assuming $d_1 \leq d_2 \leq ... \leq d_T$ without loss of generality. The following binary variables are defined for the following formulations.

$$x_{i,j} = \begin{cases} 1, & \text{if } t_i \text{ is launched by } m_j \\ 0, & \text{otherwise} \end{cases} , \forall i \in [1,T], \forall j \in [1,M]. \tag{1}$$

$$x_{i,j,h} = \begin{cases} 1, & \text{if } t_i \text{ is processed by } h^{th} \text{ core in } m_j \\ 0, & \text{otherwise} \end{cases} , \forall h \in [1,n_j], \forall i \in [1,T], \forall j \in [1,M]. \tag{2}$$

$$y_{i,k,l,h} = \begin{cases} 1, & \text{if } t_i \text{ is processed by } h^{th} \text{ core in } s_{k,l} \\ 0, & \text{otherwise} \end{cases} , \forall h \in [1,n_{k,l}], \forall i \in [1,T], \forall j \in [1,M], \forall l \in [1,s_k]. \tag{3}$$

$$z_{i,r,h} = \begin{cases} 1, & \text{if } t_i \text{ is processed by } h^{th} \text{ core in } v_r \\ 0, & \text{otherwise} \end{cases} , \forall h \in [1,n_{v_r}], \forall i \in [1,T], \forall r \in [1,V]. \tag{4}$$

This paper doesn't consider the cooperative computing between devices (Hong et al., 2019), and thus each task cannot be processed by other users' devices, i.e.,

$$x_{i,j,h} = 0, \forall h \in [1,n_j], \forall x_{i,j} = 0, \forall i \in [1,T], \quad \forall j \in [1,M]. \tag{5}$$

---

[4]The cloud resources can be provided in the form of virtual machine (VM), physical machine (PM), or both. The form of resource provisioning does not affect the application of our method.
[5]This paper considers hard deadline, where the service provider has no gains for a task if the task is finished after its deadline. The scheduling of soft deadline tasks is considered as a future work.

203   Each task can be only assigned to one core in the DE3C[6], and thus

$$\sum_{j=1}^{M}\sum_{h=1}^{n_j} x_{i,j,h} + \sum_{k=1}^{E}\sum_{l=1}^{s_k}\sum_{h=1}^{n_{k,l}} y_{i,k,l,h} + \sum_{r=1}^{V}\sum_{h=1}^{n_{v_r}} z_{i,r,h} \le 1, \forall i \in [1,T]. \tag{6}$$

204   Then the number of completed tasks is

$$N = \sum_{i=1}^{T}\left(\sum_{j=1}^{M}\sum_{h=1}^{n_j} x_{i,j,h} + \sum_{k=1}^{E}\sum_{l=1}^{s_k}\sum_{h=1}^{n_{k,l}} y_{i,k,l,h} + \sum_{r=1}^{V}\sum_{h=1}^{n_{v_r}} z_{i,r,h}\right). \tag{7}$$

When t_i is processed locally, its execution time is its computing time as there is no data transmission, i.e.,

$$\tau_i = f_i/g_j, \forall x_{i,j} = 1, \forall i \in [1,T], \forall j \in [1,M]. \tag{8}$$

205   As all tasks assigned to a core can be finished before their respective deadlines if and only if each task
206   can be finished within its deadline when they are processed in ascending order of the deadline (Pinedo,
207   2016), the finish time of each task processed by a core can be calculated by assuming tasks are processed
208   in the EDF scheme in the core. Then, if t_i is processed in the hth core of device m_j, i.e., $x_{i,j,h} = 1$, its start
209   time is the accumulated execution time of tasks that have earlier deadlines than it and are processed in the
210   same core, i.e., $\sum_{ii=1}^{i-1}(x_{i,j,h} \cdot \tau_{ii})$. Thus, the finish time of t_i can be formulated as

$$ft_{i,j,h} = x_{i,j,h} \cdot \sum_{ii=1}^{i}(x_{ii,j,h} \cdot \tau_{ii}), \forall h \in [1,n_j], \forall i \in [1,T], \forall j \in [1,M]. \tag{9}$$

211   Where $ft_{i,j,h} = 0$ when t_i is not assigned to h^{th} core of device m_j. In this situation, the deadline constraints
212   can be formulated into

$$ft_{i,j,h} \le d_i, \forall h \in [1,n_j], \forall i \in [1,T], \forall j \in [1,M]. \tag{10}$$

213   When a task is offloaded to an edge or the cloud, its execution time is formed from the data transfer time
214   and computing time. If a task is scheduled to a core of an edge server or a cloud server, its computing can
215   be started only when its data transmission is complete and the core has finished all tasks that have earlier
216   deadlines and are assigned to the core (recall that EDF scheduling provides the best solution for a core
217   in SLA optimization). The earliest completion time of data transmission for a task offloaded to an edge
218   server or the cloud is

$$dt_{i,k,l,h} = y_{i,k,l,h} \cdot \sum_{ii=1}^{i}\left(y_{ii,k,l,h} \cdot \frac{in_{ii}}{b_{j,k,l}}\right), \forall h \in [1,n_{k,l}], \forall i \in [1,T], \forall j \in [1,M], \forall l \in [1,s_k], \tag{11}$$

219   or

$$dt_{i,v_r,h} = z_{i,r,h} \cdot \sum_{ii=1}^{i}\left(z_{ii,r,h} \cdot \frac{in_{ii}}{b_r}\right), \forall h \in [1,n_{v_r}], \forall i \in [1,T], \forall r \in [1,V]. \tag{12}$$

220   And the ready time of a core for computing an offloaded task in an edge or the cloud is the latest finish
221   time of tasks that are offloaded to the same core and have earlier deadlines, which can be formulated into

$$rt_{i,k,l,h} = y_{i,k,l,h} \cdot \max_{ii=1}^{i-1}(y_{ii,k,l,h} \cdot ft_{ii,k,l,h}), \forall h \in [1,n_{k,l}], \forall i \in [1,T], \forall j \in [1,M], \forall l \in [1,s_k], \tag{13}$$

222   or

$$rt_{i,v_r,h} = z_{i,r,h} \cdot \sum_{ii=1}^{i-1}(z_{ii,r,h} \cdot ft_{ii,v_r,h}), \forall h \in [1,n_{v_r}], \forall i \in [1,T], \forall r \in [1,V], \tag{14}$$

223   where $ft_{i,k,l,h}$ and $ft_{i,v_r,h}$ represent the finish time of task t_i when it is assigned to h^{th} core in the edge
224   server $s_{k,l}$ and the cloud server v_r respectively. For a core processing an offloaded task, its start time is a

---

[6]In this paper, the redundant execution for a task is not employed to improve the task performance (Liu et al., 2019a) as it cost more resources. Each task is assumed to be processed by only one core, as done in many published articles. A task executed on more than one cores usually can be decomposed into several subtasks. One is suggested to refer to our previous work (Wang et al., 2019a) which studied on the task scheduling with parallelism awareness, which complements this work.

later of its ready time and the completion time of the input data[7]. Thus, the finish time of offloaded tasks can be calculated as followings

$$ft_{i,k,l,h} = y_{i,k,l,h} \cdot (\max\{dt_{i,k,l,h}, rt_{i,k,l,h}\} + \frac{f_i}{g_{k,l}}), \forall h \in [1, n_{k,l}], i \in [1, T], \forall l \in [1, s_k], \forall k \in [1, E]. \quad (15)$$

$$ft_{i,v_r,h} = z_{i,r,h} \cdot (\max\{dt_{i,v_r,h}, rt_{i,v_r,h}\} + \frac{f_i}{g_{v_r}}), \forall h \in [1, n_{v_r}], \forall i \in [1, T], \forall r \in [1, V]. \quad (16)$$

And the deadline constraints of task processing in edges and the cloud can be formulated into

$$ft_{i,k,l,h} \le d_i, \forall h \in [1, n_{k,l}], i \in [1, T], \forall j \in [1, M], \forall l \in [1, s_k]. \quad (17)$$

$$ft_{i,v_r,h} \le d_i, \forall h \in [1, n_{v_r}], \forall i \in [1, T], \forall r \in [1, V]. \quad (18)$$

For a computing node (a device, an edge server or a cloud server), the time occupied for processing tasks is the latest finish time of all tasks assigned to the node, i.e.,

$$ut_{m_j} = \max_{i=1}^{T} \max_{h=1}^{n_j} ft_{i,j,h}, \forall j \in [1, M], \quad (19)$$

$$ut_{s_{k,l}} = \max_{i=1}^{T} \max_{h=1}^{n_{k,l}} ft_{i,k,l,h}, \forall l \in [1, s_k], \forall k \in [1, E], \quad (20)$$

$$ut_{v_r} = \max_{i=1}^{T} \max_{h=1}^{n_{v_r}} ft_{i,v_r,h}, \forall r \in [1, V], \quad (21)$$

where $ut_{\mathcal{N}}$ is the use time of the computing node $\mathcal{N}$ for finishing tasks assigned to it. Then, the occupied resource amount ($or_{\mathcal{N}}$) of a computing node for task processing are respectively

$$or_{m_j} = ut_{m_j} \cdot g_j \cdot n_j, \forall j \in [1, M], \quad (22)$$

$$or_{s_{k,l}} = ut_{s_{k,l}} \cdot g_{k,l} \cdot n_{k,l}, \forall l \in [1, s_k], \forall k \in [1, E], \quad (23)$$

$$or_{v_r} = ut_{v_r} \cdot g_{v_r} \cdot n_{v_r}, \forall r \in [1, V], \quad (24)$$

And the consumed computing resource amount $cr_{\mathcal{N}}$ can be quantified by the accumulated processing length of its finished tasks for each computing node, i.e.,

$$cr_{m_j} = \sum_{i=1}^{T} \sum_{h=1}^{n_j} x_{i,j,h} \cdot f_i, \forall j \in [1, M], \quad (25)$$

$$cr_{s_{k,l}} = \sum_{i=1}^{T} \sum_{h=1}^{n_{k,l}} y_{i,k,l,h} \cdot f_i, \forall l \in [1, s_k], \forall k \in [1, E], \quad (26)$$

$$cr_{v_r} = \sum_{i=1}^{T} \sum_{h=1}^{n_{v_r}} z_{i,r,h} \cdot f_i, \forall r \in [1, V], \quad (27)$$

Thus, the computing resource utilization of devices, edge servers, and cloud servers are respectively

$$U_{device} = \frac{\sum_{j=1}^{M} cr_{m_j}}{\sum_{j=1}^{M} or_{m_j}}, \quad (28)$$

---

[7]Usually the amount of result data is very small, and its transmission time is negligible compared with that of input data or the computing time. Thus, in this paper, the transmission time of the result data is ignored, as done by many previous works, such as (Zhao and Zhou, 2019; Chen and Hao, 2018; Hong et al., 2019).

243

$$U_{edge} = \frac{\sum_{k=1}^{E} \sum_{l=1}^{s_k} cr_{s_{k,l}}}{\sum_{k=1}^{E} \sum_{l=1}^{s_k} or_{s_{k,l}}}, \tag{29}$$

244

$$U_{cloud} = \frac{\sum_{r=1}^{V} cr_{v_r}}{\sum_{r=1}^{V} or_{v_r}}, \tag{30}$$

245 and the overall resource utilization of the DE3C system is

$$U = \frac{\sum_{j=1}^{M} cr_{m_j} + \sum_{k=1}^{E} \sum_{l=1}^{s_k} cr_{s_{k,l}} + \sum_{r=1}^{V} cr_{v_r}}{\sum_{j=1}^{M} or_{m_j} + \sum_{k=1}^{E} \sum_{l=1}^{s_k} or_{s_{k,l}} + \sum_{r=1}^{V} or_{v_r}}. \tag{31}$$

Based on the above formulation, the task scheduling problem optimizing the SLA satisfaction can be modelled as

Maximizing $N + U$ (32)

246 subject to

$$(1) - (31) \tag{33}$$

247 where the objective (32) is to maximize the number of finished tasks, and to maximize the overall
248 computing resource utilization when the finished task number cannot be improved. The decision variables
249 include $x_{i,j,h}$ ($h \in [1, n_j]$, $j \in [1, M]$, $i \in [1, T]$), $y_{i,k,l,h}$ ($h \in [1, n_{k,l}]$, $l \in [1, s_k]$, $k \in [1, E]$, $i \in [1, T]$), and
250 $z_{i,r,h}$ ($h \in [1, n_{v_r}]$, $r \in [1, V]$, $i \in [1, T]$). This problem is binary nonlinear programming (BNLP), which can
251 be solved by existed tools, e.g., lp_solve (Berkelaar et al., 2020). While these tools are implemented based
252 on branch and bound, which is not applicable to large-scale problems. Therefore, a heuristic method is
253 proposed to solve the problem in polynomial time in the next section.

THREE-STAGE HEURISTIC TASK SCHEDULING

255 This section presents the proposed hybrid heuristic method, called TSSLA (Three-Stage scheduling
256 optimizing SLA), to address the task scheduling problem stated in the previous section, which coordinates
257 the richness of cloud computing resources and the low transmission delay of local and edge resources, to
258 optimize the SLA satisfaction.

259 The proposed hybrid heuristic method includes three stages, where the first stage tries to satisfy
260 deadlines of as many tasks as possible, by prioritising the usage of abundant cloud computing resources
261 and assigning tasks that cannot be finished in the cloud to edges or corresponding devices. In the second
262 stage, TSSLA tries to make full use of local resources and release some edge resources by rescheduling
263 several tasks from edges to corresponding devices, and to exploit edge resources shared by multiple users
264 for finishing more tasks. At the last stage, our method aims at optimizing the cost of cloud resources, by
265 rescheduling as many tasks as possible from the cloud to corresponding devices and edges. Algorithm 1
266 outlines the proposed three-stage hybrid heuristic task scheduling.

267 As shown in Algorithm 1, in the first stage, TSSLA first pre-assigns all tasks that can be finished by
268 cloud resources to the cloud (line 1 in Algorithm 1). For each task, TSSLA pre-rents a one-core VM
269 instance with the best cost performance. In real world, a public cloud, e.g., Amazon EC2 (Amazon, 2020),
270 provides various VM types configured with different core numbers, and for each type [8], e.g., c6g.* in
271 Amazon EC2, VM instances have a same price per core. Thus, TSSLA pre-rents one-core VM instances
272 in this stage. After this step, there is no task can be finished in the cloud, and TSSLA pre-assigns remain
273 tasks to edge servers employing LASTF and EDF for computing core selection and task ordering in each
274 core, respectively (see Algorithm 2). Then, TSSLA schedules tasks to each device adopting Algorithm 2.

275 At the second stage, TSSLA examines each task assigned to edges. If the task can be finished in its
276 device, TSSLA reassigns the task to the device, and the edge has more available resources for processing
277 unassigned tasks. Thus, after that, TSSLA repeats step 2 in the first stage, which assigns remain tasks to

---

[8]This paper assumes that cloud resources are provided on demand, and leave the concern of other provisioning scheme, e.g., spot, as a future work.

Algorithm 1 The three-stage hybrid heuristic task scheduling

Input: the information of tasks and DE3C resources
Output: the mapping between tasks and computing resources
  //The first stage
1: iteratively pre-assigning a task to a new cloud server core, until no task can be finished in the cloud;
2: iteratively pre-assigning a task to an edge server core using Algorithm 2, until no task can be finished in edges;
3: iteratively assigning a task to a device core using Algorithm 2, until no task can be finished locally;
  //The second stage
4: for all tasks assigned to edges, iteratively re-assigning a task to a device core using Algorithm 2, until no task can be finished locally;
5: for remaining unassigned tasks, iteratively assigning a task to an edge server core using Algorithm 2, until no task can be finished in edges;
  //The third stage
6: for all tasks assigned to the cloud, iteratively re-assigning a task to a device core using Algorithm 2, until no task can be finished locally;
7: for all tasks assigned to the cloud, iteratively re-assigning a task to an edge server core using Algorithm 2, until no task can be finished locally;
8: for tasks assigned to the cloud, iteratively re-assigning the task in the next cloud VM to one of previous VMs using Algorithm 2, and de-renting idle cloud VMs;

---

edge servers by Algorithm 2. Now, no more tasks can be finished by DE3C resources, thus TSSLA only improves the resource usage in the last stage.

TSSLA employs two approaches to improve the resource efficiency in the third stage. One approach is trying to reassign as many tasks as possible from the cloud to local devices and edge servers because local resources and edge server resources are cheaper and have much less network latency, compared with cloud resources. And another is to consolidate tasks assigned to the cloud for improving the cost efficiency by reducing the idle time of VM instances, as cloud resources are charged by time unit, e.g., hour.

Therefore, in the third stage, TSSLA examines each task assigned to the cloud, and if the task can be finished by corresponding device, TSSLA reassigns the task to the device (see line 6 in Algorithm 1). Otherwise, TSSLA checks whether the task's requirements can be satisfied by an edge server, and if so reassigns the task to the edge server (see line 7 in Algorithm 1). After these reassignments, TSSLA reassigns the task that has assigned to the next VM to one of its previous VMs by using Algorithm 2, and re-rents idle VMs. This step reduces the idle time of VMs, which improves the cost efficiency, as the rent time of each VM is round up to times of the charge unit for its cost. For example, if one user rents a VM for 1.8 hours, and the cloud provider charges \$0.1 per hours, the user must pay \$0.2 (\$0.1/hours $\times \lceil 1.8 \rceil$ hours) for the VM.

TSSLA employs Algorithm 2 to implement all of these above task assignments in each device, edges, and the cloud, which decides which core to process the task. The detail is shown in the following.

As shown in Algorithm 2, to select an available computing core for a task, TSSLA traverses each available core (line 2 in Algorithm 2), and calculates the accumulated slack time with the assumptions that the task is assigned to the core and all tasks are executed in the ascending order of their deadlines (lines 3–8 in Algorithm 2). Then, TSSLA allocates the core providing LAST to the task (lines 9–15 in Algorithm 2). For each available core, if the assignment of the task results in at least one deadline violation (line 4 in Algorithm 2), Algorithm 2 returns false which means the requirements of the task cannot be satisfied by any of these available computing cores (line 17 in Algorithm 2).

RESULTS

This section conducts simulated experiments designed by referring to related works and real worlds, to evaluate the performance of the proposed method.

Experiment Design

A DE3C system is established, which is composed of one public cloud, an edge, and 10 user devices. The set of various system parameters are referring to that of (Du et al., 2019; Chen et al., 2016; Alkhalaileh et al., 2020), the Gaia Cluster (University of Luxembourg, 2020), as well as Amazon EC2 (Amazon,

---

Algorithm 2 Scheduling a task in a device, edges or the cloud

---

Input: the information of the task (t), computing cores and tasks assigned to each core
Output: the computing core the task is assigned to, and **False** if the task cannot be finished by any core

1: $LAST \leftarrow +\infty$;//recording the least accumulated slack time
2: **for** each core **do**
3:     assuming t is assigned to the core, calculating the finish time of each task
       assigned to the core, using (8) and (9) for device cores, (11), (13) and (15)
       for edge server cores, and (12), (14) and (16) for cloud VMs;
4:     **if** the finish time of one task is later than its deadline **then**
5:         **continue**;
6:     **end if**
7:     calculating the slack time of each task, which is the difference between its finish
       time and its deadline;
8:     accumulating each slack time of all tasks;
9:     **if** the accumulated slack time $< LAST$ **then**
10:         $LAST \leftarrow$ the accumulated slack time;
11:         recording the core;
12:     **end if**
13: **end for**
14: **if** $LAST < +\infty$ **then**
15:     **return** the last recorded core;
16: **else**
17:     **return False**;
18: **end if**

---

311 2020), which are detailed as followings and shown in Table 2. 100 tasks are generated randomly. Each
312 task is randomly associated to one device that is regarded as the device launching the task. The length and
313 the size of each task are randomly set in ranges of [1, 2000]GHz and [20, 500]MB, respectively, to cover
314 small to large tasks. The computing capacities of each core in a device, an edge server, a cloud VM are
315 randomly in the ranges [1, 2]GHz, [2, 3]GHz, and [2, 3]GHz, respectively. The number of computing
316 core is randomly set in the ranges [1, 4] and [4, 8], respectively for each device and each edge server. The
317 number of servers is set in [1, 4] for the edge. The price of each core is \$0.01 per hour for cloud VMs.
318 The bandwidths for transmitting data from a device to the edge and the cloud are in [10, 100]Mbps and [1,
319 10]Mbps respectively.

Table 2. The parameters of simulated DE3C system.

Tasks				Device	Edge Server	Cloud VM
Number	100		Number	10	[1, 4]	-
Computing length	[1, 2000]GHz		Core number	[1, 4]	[4, 8]	-
Processed data size	[20, 500]MB		Capacity per core	[1, 2]GHz	[2, 3]GHz	[2, 3]GHz
Deadline	[500, 1500]s		Bandwidth	-	[10, 100]Mbps	[1, 10]Mbps
			Price	-	-	\$0.01/hour

320   The performance of TSSLA is compared with the following classical or state-of-the-art scheduling
321 methods designed for DE3C system. As done in all of the existed works (the best of our knowledge), the
322 following methods employ local resources first, and the cloud resource at last, for processing tasks.

- **FF** (First Fit) (Bays, 1977) iteratively assigns the first task to the first core satisfying its requirements.

- **FFD** (First Fit Decreasing) (B.V. and Guddeti, 2018) iteratively assigns the largest task to the first
  core satisfying its requirements.

- **EDF** (Earliest Deadline First) (Benoit et al., 2021) iteratively assigns the task with the earliest
  deadline to the first core satisfying its requirements.

- **BF** (Best Fit) (Zhao and Kim, 2020) iteratively assigns the first task to the core that satisfies its
  requirements and provides the latest finish time for the task.

- **EFTF** (Earliest Finish Time First) which is the basic idea of the method proposed by Liu et al. (Liu et al., 2019a), iteratively assigns the first task to the core providing the earliest finish time.

- **EDF_EFTF**, the idea of Stavrinides's and Karatza's proposed method (Stavrinides and Karatza, 2019), respectively employs EDF and EFTF for task selection and resource selection.

- **LSTF** (Least Slack Time First) (Michel et al., 2021) iteratively assigns the first task to the core providing the least slack time.

- **LSSRF** (Least Size-Slack time ratio First), the idea employed by Mahmud et al. (Mahmud et al., 2020), iteratively assigns the first task to the core providing the maximal ratio between the profit and the slack time. The length is regarded as the profit for each task.

The performance metrics used to quantify the performance of each task scheduling method include the followings.

- **SLA satisfaction** can be quantified by the amount of finished tasks in number, length, and processed data size, which are respectively calculated by Eq. (7), (34), and (35). The larger value is better for the metric. For the length and the processed data size of finished tasks, the followings report the percentages of that of all launched tasks.

$$len = \sum_{i=1}^{T}((\sum_{j=1}^{M}\sum_{h=1}^{n_j}x_{i,j,h}+\sum_{k=1}^{E}\sum_{l=1}^{s_k}\sum_{h=1}^{n_{k,l}}y_{i,k,l,h}+\sum_{r=1}^{V}\sum_{h=1}^{n_{v_r}}z_{i,r,h})\cdot f_i). \qquad (34)$$

$$size = \sum_{i=1}^{T}((\sum_{j=1}^{M}\sum_{h=1}^{n_j}x_{i,j,h}+\sum_{k=1}^{E}\sum_{l=1}^{s_k}\sum_{h=1}^{n_{k,l}}y_{i,k,l,h}+\sum_{r=1}^{V}\sum_{h=1}^{n_{v_r}}z_{i,r,h})\cdot in_i). \qquad (35)$$

- **Resource utilization** is one of the most popular metrics to quantify the resource efficiency, which is the ratio between amounts of consumed resources and occupied resources, i.e., U calculated by Eq. (31). It is better for a higher value.

- **Makespan** is the latest finish time of tasks, which can be achieved by Eq. 36. Earlier makespan means faster processing rate, and thus is better.

$$makespan = \max_{i=1}^{T}\{\max\{\max_{j=1}^{M}\max_{h=1}^{n_j}ft_{i,j,h},\max_{k=1}^{E}\max_{l=1}^{s_k}\max_{h=1}^{n_{k,l}}ft_{i,k,l,h},\max_{r=1}^{V}\max_{h=1}^{n_{v_r}}ft_{i,v_r,h}\}\}. \qquad (36)$$

- **Cost efficiency** is the length of tasks processed by per-dollar resource in the cloud, as calculated by Eq. (37). It is a metric for quantifying the resource efficiency in clouds. A greater value is better.

$$Ceff = \frac{len}{\sum_{r=1}^{V}\lceil cr_{v_r}\cdot p_r\rceil} \qquad (37)$$

Experiment Results

SLA Satisfaction

Fig. 2 shows the performance of various task scheduling methods in SLA satisfaction. As shown in the figure, TSSLA has 22.2%–27.6%, 47.3%–59.1%, and 25.4%–32.6% better SLA satisfaction performance compared with other methods in task number, task computing length, and processed data size, respectively. The superiority of our method is allocating computing resources according to the scarcity degree of resources. TSSLA prefers using the abundant computing resources of the cloud, and employs scarce computing resources of edges and devices for processing tasks cannot be finished by the cloud due to the poor network performance between users and their cloud, in its first stage. In contrary, other methods prefer local resources or nearby edge resources, aiming at providing the best performance for each task with minimal resource costs. But these methods result in several local and nearby edge resources that are used by some tasks which can be finished by the cloud, and these resources can be reserved for processing other tasks whose demands cannot be satisfied by the cloud. Thus, our method has a better performance than other methods in SLA satisfaction. Based on the idea of our method, these works can be improved by

(a) SLA in number

(b) SLA in computing length

(c) SLA in data size
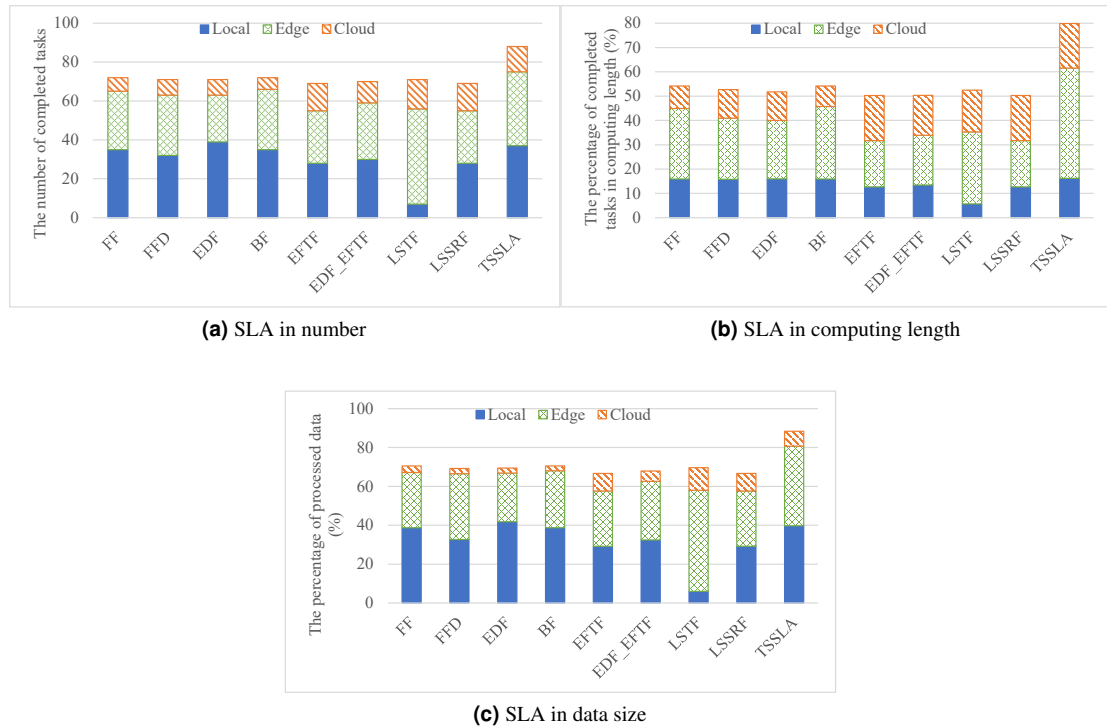
Figure 2. The SLA satisfaction performance in task number, task computing length, and processed data size respectively, achieved by various scheduling methods.

reassigning some tasks from local devices or edges to the cloud, to make some resources idle for finishing remaining unassigned tasks.

Fig. 2 also shows that, except TSSLA, EDF has the best performance in optimizing SLA satisfaction in devices, and LSTF achieves the most number of completed tasks in the edge. Even so, all methods except TSSLA have comparable performance in SLA optimization overall. The reason why EDF is better than FF, FFD, BF and EFTF in SLA optimization in the devices is that EDF prioritizes the demands of tasks with tight deadlines, and thus postpones tasks with more slack time, which can finish more tasks with tight deadlines compared with other methods. Besides, EDF yields an optimal schedule for maximizing the number of finished tasks in each core (Pinedo, 2016). While, after completing more tasks locally, there are fewer tasks can be finished in edges or the cloud, as shown in Fig. 2, when applying EDF. This phenomenon doesn't occur when employing TSSLA. TSSLA satisfies more tasks not only in local devices but also in the edge, compared with other methods (except EDF in devices and LSFT in the edge). This is mainly because TSSLA assigns tasks that can be finished by both the cloud and local devices or the edge to the cloud at the first, which results in more available local and edge resources for processing tasks whose demands can be satisfied only by local devices or the edge. This further verifies the high efficiency of our method. The main reason why LSTF provides the most number of completed tasks at the edge is that it completes much fewer tasks by local resources, compared with other methods, and thus leaves more tasks with loose deadlines to the edge for processing.

Resource Utilization

As shown in Fig. 3, TSSLA achieves almost same overall resource utilization to BF, and 4.1%–54.6% higher than other methods, which verifies that our method provides high resource efficiency for task processing in DE3C environments. The reason that TSSLA achieves a higher overall resource utilization, compared with other methods, is because it provides a much better utilization than others in the edge, as shown in Fig. 3-c. In addition, TSSLA completes more than half of tasks' computing length at the edge, as shown in Fig. 2-b. In general, more tasks processed by the edge or the cloud means higher resource utilization in the edge or the cloud (shown in Fig. 2, Fig. 3-c, and Fig. 3-d). This is because more tasks can result in less ratio between the amounts of idle computing resources and occupied computing

385 resources, as the data transmission and the data computing can be parallel for different tasks, in each
386 core. While, compared with LSTF, TSSLA achieves 41.7% higher utilization, as shown in Fig. 3-c,
387 although it completes 22.5%, 13.3%, and 21.1% less tasks in number, computing length, and processed
388 data size, respectively, as shown in Fig. 2, in the edge. The main reason is that TSSLA has a greater ratio
389 between the computing length and the processed data size of tasks completed by the edge than LSTF,
390 which leads to less idle computing resources, and thus results in a higher computing resource utilization.
391 This phenomena can be exploited to design heuristic scheduling methods for optimizing the efficiency of
392 computing resources in edges and clouds.
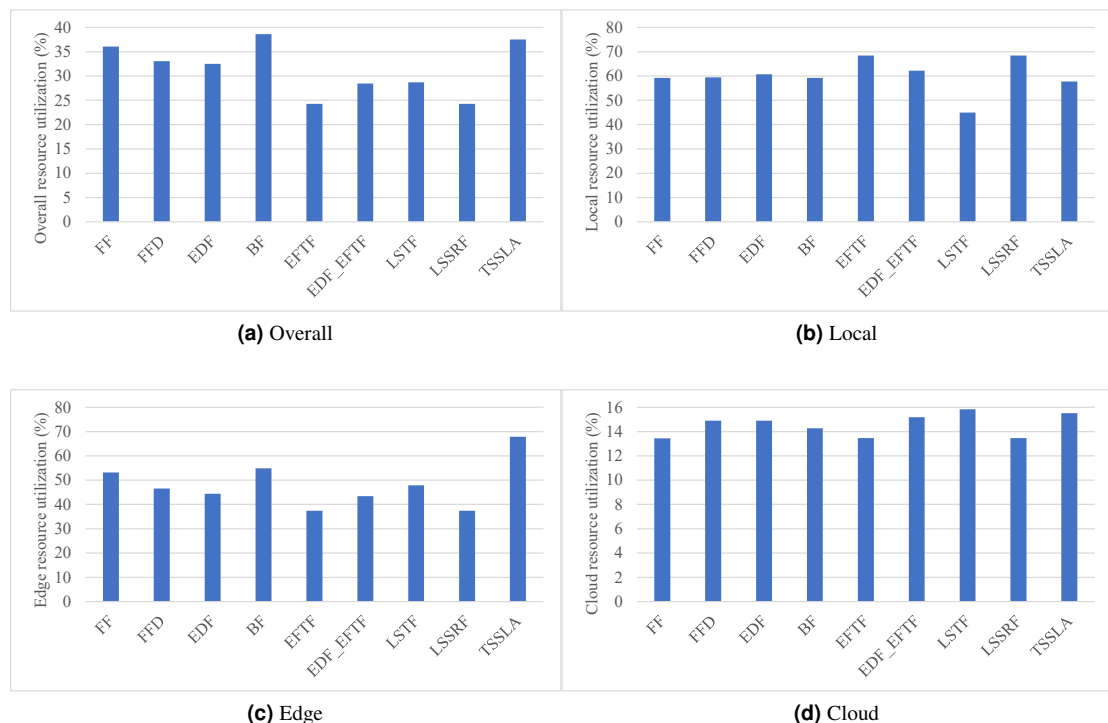
Figure 3. The overall computing resources of the DE3C, devices, edge servers, and cloud servers, when applying various scheduling methods.

393 The reason why BF achieves the highest overall resource utilization, as shown in Fig. 3-a, is that
394 the tasks processed by the cloud is the least in each SLA metric, as shown in Fig. 2, and thus the low
395 utilization of cloud computing resources has the lowest impact on the overall resource utilization.

396 The utilization of cloud computing resources is much less than that of edge computing resources
397 for each method, as shown in Fig. 3-d, which is because the network performance of the cloud is much
398 worse, leading more idle computing resources due to the longer time of data transmission, compared
399 with that of the edge. Thus, it would be good to assign tasks with small data sizes to the cloud. While
400 tasks processing less data usually have larger computing length, and thus there is a trade-off between the
401 limited computing resources of devices or edges and the poor network performance of the cloud, which is
402 one of our considerations for designing highly efficient scheduling methods in future.

403 ***Makespan***
404 Fig. 4 shows the makespan when applying various scheduling methods. As shown in the figure, TSSLA
405 has a larger makespan than other methods except LSTF, as the DE3C completes the most computing
406 length and processes the most data of tasks when applying TSSLA, and the makespan is usually increased
407 with the completed computing length and the processed data size. In fact, TSSLA has only about 20%
408 larger makespan, but it completes more than 47.3% more computing length and processes more than
409 25.4% more data than other methods except LSTF. In addition, LSTF has larger makespan to TSSLA,
410 even though it completes much less computing length and processes much fewer data than TSSLA. These
411 results validate the efficiency of our methods further.
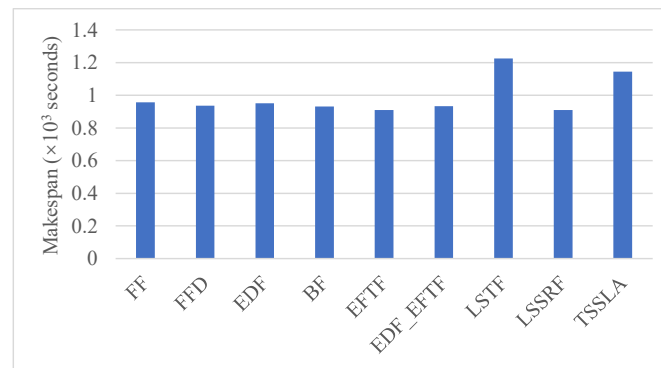
Figure 4. The latest finish time of tasks in the DE3C, when applying various scheduling methods.
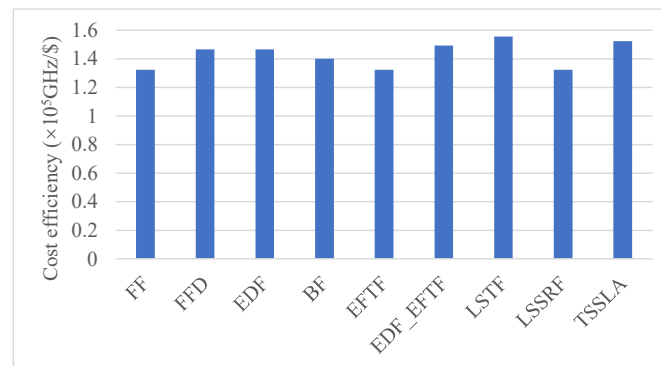
Figure 5. The cost efficiency of cloud computing resources, when applying various scheduling methods.

412   The major reason of LSTF having the largest makespan is that it completes the most number of tasks
413 in both the edge and the cloud, and the local device provides much less processing time than an edge and
414 especially the cloud for a task due to the data transmission time from the device to the edge and the cloud.

415   *Cost Efficiency*
416 Fig. 5 shows the cost efficiency for task processing in the cloud. TSSLA has a comparable cost efficiency
417 to LSTF and is better than others. Comparing Fig. 3-d and Fig. 5, we can see that the relative performance
418 of the cost efficiency is almost same as that of the computing resource utilization. This is mainly because
419 the cloud computing resources are charged by the use time. Thus, in most time, the resource utilization
420 and the cost efficiency are equivalent for quantifying the usage efficiency of cloud computing resources.
421 The reason why LSTF has a good cost efficiency is that it completes the most number of tasks in the
422 cloud, as illustrated in section .

CONCLUSIONS

424 This paper studies on the SLA satisfaction optimization in a device-edge-cloud cooperative computing
425 (DE3C) environment. This paper first formulates the problem into a BNLP, and then proposes a heuristic
426 scheduling method, named TSSLA, to solve the problem in polynomial-time complexity. TSSLA consists
427 of three heuristic stages which respectively exploit the abundant computing resources of the cloud, the
428 shared resources of edges, and the low/zero network latency of edge and device resources, for optimizing
429 the number of tasks whose requirements are satisfied and the resource efficiency. Experiment results
430 confirm the superior performance of TSSLA in optimizing SLA satisfaction and resource efficiency.

431   In fact, our method improves the SLA satisfaction and the resource efficiency by improving the
432 collaboration ability among devices, edges, and clouds to exploit all of their benefits. This idea can be
433 also applied to other hybrid computing systems, e.g., multi-clouds, hybrid clouds, which is one of our
434 future work.

435   This paper focuses on the task scheduling for DE3C environments, assuming the data is transmitted to
436 the computing node only when the offloading decision is made for each task. Caching data in edge servers

and especially the cloud in advance can improve the performance of task executions. Thus, the prediction of offloading decisions and the caching strategy in DE3C will be studied in the future. In addition, the design of cache-aware task scheduling methods will be concerned to improve the benefits of caching strategies.

REFERENCES

Alkhalaileh, M., Calheiros, R. N., Nguyen, Q. V., and Javadi, B. (2020). Data-intensive application scheduling on mobile edge cloud computing. *Journal of Network and Computer Applications*, 167:102735.

Amazon (2020). Amazon web services – cloud computing services. https://aws.amazon.com/.

Apat, H. K., s. Compt, B., Bhaisare, K., and Maiti, P. (2019). An optimal task scheduling towards minimized cost and response time in fog computing infrastructure. In *2019 International Conference on Information Technology (ICIT)*, pages 160–165.

Balasubramanian, V., Otoum, S., Aloqaily, M., Al Ridhawi, I., and Jararweh, Y. (2020). Low-latency vehicular edge: A vehicular infrastructure model for 5g. *Simulation Modelling Practice and Theory*, 98:101968.

Bays, C. (1977). A comparison of next-fit, first-fit, and best-fit. *Commun. ACM*, 20(3):191–192.

Benoit, A., Elghazi, R., and Robert, Y. (2021). Max-stretch minimization on an edge-cloud platform. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 766–775.

Berkelaar, M., Dirks, J., Eikland, K., Notebaert, P., Ebert, J., and Gourvest, H. (2020). lpsolve: A mixed integer linear programming (milp) solver. http://sourceforge.net/projects/lpsolve.

B.V., N. and Guddeti, R. M. R. (2018). Heuristic-based iot application modules placement in the fog-cloud computing environment. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 24–25.

Chen, L., Guo, K., Fan, G., Wang, C., and Song, S. (2020). Resource constrained profit optimization method for task scheduling in edge cloud. *IEEE Access*, 8:118638–118652.

Chen, M. and Hao, Y. (2018). Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597.

Chen, X., Jiao, L., Li, W., and Fu, X. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808.

Chen, X., Zhang, J., Lin, B., Chen, Z., Wolter, K., and Min, G. (2022). Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):683–697.

Chen, Y., Zhang, N., Zhang, Y., and Chen, X. (2019). Dynamic computation offloading in edge computing for internet of things. *IEEE Internet of Things Journal*, 6(3):4242–4251.

Cisco (2020). Cisco annual internet report (2018–2023). https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html.

Du, J., Zhao, L., Chu, X., Yu, F. R., Feng, J., and I, C. L. (2019). Enabling low-latency applications in lte-a based mixed fog/cloud computing systems. *IEEE Transactions on Vehicular Technology*, 68(2):1757–1771.

Gao, G., Xiao, M., Wu, J., Huang, H., Wang, S., and Chen, G. (2019). Auction-based vm allocation for deadline-sensitive tasks in distributed edge cloud. *IEEE Transactions on Services Computing (Early Access)*, pages 1–14.

Ghasempour, A. (2019). Internet of things in smart grid: Architecture, applications, services, key technologies, and challenges. *Inventions*, 4(1):1–22.

Han, Z., Tan, H., Li, X., Jiang, S. H. ., Li, Y., and Lau, F. C. M. (2019). Ondisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds. *IEEE/ACM Transactions on Networking*, 27(6):2472–2485.

Hong, Z., Chen, W., Huang, H., Guo, S., and Zheng, Z. (2019). Multi-hop cooperative computation offloading for industrial iot-edge-cloud computing environments. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2759–2774.

Islam, A., Debnath, A., Ghose, M., and Chakraborty, S. (2021). A survey on task offloading in multi-access edge computing. *Journal of Systems Architecture*, 118:102225.

Kumar, M., Sharma, S., Goel, A., and Singh, S. (2019). A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 143:1 – 33.

Lakhan, A. and Li, X. (2019). Content aware task scheduling framework for mobile work-

flow applications in heterogeneous mobile-edge-cloud paradigms: Catsa framework. In *2019 ISPA/BDCloud/SocialCom/SustainCom*, pages 242–249.

Li, C., Wang, C., and Luo, Y. (2020). An efficient scheduling optimization strategy for improving consistency maintenance in edge cloud environment. *The Journal of Supercomputing*, 76:6941–6968.

Liang, D. and Xu, Z. (2017). The new extension of topsis method for multiple criteria decision making with hesitant pythagorean fuzzy sets. *Applied Soft Computing*, 60:167 – 179.

Liu, L., Tan, H., Jiang, S. H.-C., Han, Z., Li, X.-Y., and Huang, H. (2019a). Dependent task placement and scheduling with function configuration in edge computing. In *Proceedings of the International Symposium on Quality of Service*, IWQoS'19, pages 1–10, New York, NY, USA. Association for Computing Machinery.

Liu, Y., Yang, C., Jiang, L., Xie, S., and Zhang, Y. (2019b). Intelligent edge computing for iot-based energy management in smart cities. *IEEE Network*, 33(2):111–117.

Ma, Z., Zhang, S., Chen, Z., Han, T., Qian, Z., Xiao, M., Chen, N., Wu, J., and Lu, S. (2022). Towards revenue-driven multi-user online task offloading in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(5):1185–1198.

Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. (2020). Profit-aware application placement for integrated fog–cloud computing environments. *Journal of Parallel and Distributed Computing*, 135:177 – 190.

Meng, J., Tan, H., Li, X., Han, Z., and Li, B. (2020). Online deadline-aware task dispatching and scheduling in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1270–1286.

Meng, J., Tan, H., Xu, C., Cao, W., Liu, L., and Li, B. (2019). Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2287–2295.

Miao, Y., Wu, G., Li, M., Ghoneim, A., Al-Rakhami, M., and Hossain, M. S. (2020). Intelligent task prediction and computation offloading based on mobile-edge cloud computing. *Future Generation Computer Systems*, 102:925 – 931.

Michel, O., Bifulco, R., Rétvári, G., and Schmid, S. (2021). The programmable data plane: Abstractions, architectures, algorithms, and applications. *ACM Comput. Surv.*, 54(4).

Papadakis-Vlachopapadopoulos, K., González, R. S., Dimolitsas, I., Dechouniotis, D., Ferrer, A. J., and Papavassiliou, S. (2019). Collaborative sla and reputation-based trust management in cloud federations. *Future Generation Computer Systems*, 100:498–512.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*, chapter 2, pages 13–32. Springer, 5 edition.

Ren, J., Yu, G., He, Y., and Li, G. Y. (2019). Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5):5031–5044.

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1(1):83–98.

Sorrel, S. (2018). The internet of things: Consumer industrial & public services 2018–2023. https://www.juniperresearch.com/press/press-releases/iot-connections-to-grow-140-to-hit-50-billion.

Stavrinides, G. L. and Karatza, H. D. (2019). A hybrid approach to scheduling real-time iot workflows in fog and cloud environments. *Multimedia Tools and Applications*, 78(17):24639–24655.

Strumberger, I., Bacanin, N., Tuba, M., and Tuba, E. (2019). Resource scheduling in cloud computing based on a hybridized whale optimization algorithm. *Applied Sciences*, 9(22).

University of Luxembourg (2020). The gaia cluster - hpc @ uni.lu. https://hpc.uni.lu/systems/gaia/.

Wang, B., Song, Y., Cao, J., Cui, X., and Zhang, L. (2019a). Improving task scheduling with parallelism awareness in heterogeneous computational environments. *Future Generation Computer Systems*, 94:419–429.

Wang, B., Song, Y., Wang, C., Huang, W., and Qin, X. (2020a). A study on heuristic task scheduling optimizing task deadline violations in heterogeneous computational environments. *IEEE Access*, 8:205635–205645.

Wang, B., Wang, C., Huang, W., Song, Y., and Qin, X. (2020b). A survey and taxonomy on task offloading for edge-cloud computing. *IEEE Access*, 8:186080–186101.

Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A., and Georgalas, N. (2021). Dependent task offloading for edge computing based on deep reinforcement learning. *IEEE Transactions on Computers (In Press)*,

546   pages 1–13.

547   Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z., and Mohapatra, P. (2019b). Edge cloud offloading
548   algorithms: Issues, methods, and perspectives. *ACM Computing Surveys*, 52(1).

549   Wu, C.-J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., Hazelwood, K., Isaac, E., Jia,
550   Y., Jia, B., Leyvand, T., Lu, H., Lu, Y., Qiao, L., Reagen, B., Spisak, J., Sun, F., Tulloch, A., Vajda,
551   P., Wang, X., Wang, Y., Wasti, B., Wu, Y., Xian, R., Yoo, S., and Zhang, P. (2019). Machine learning
552   at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High
553   Performance Computer Architecture (HPCA)*, pages 331–344. IEEE.

554   You, C., Huang, K., Chae, H., and Kim, B. (2017). Energy-efficient resource allocation for mobile-edge
555   computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411.

556   Yuan, H. and Zhou, M. (2020). Profit-maximized collaborative computation offloading and resource
557   allocation in distributed cloud and edge computing systems. *IEEE Transactions on Automation Science
558   and Engineering (In press)*, pages 1–11.

559   Zhang, C., Du, H., Ye, Q., Liu, C., and Yuan, H. (2019). Dmra: A decentralized resource allocation scheme
560   for multi-sp mobile edge computing. In *2019 IEEE 39th International Conference on Distributed
561   Computing Systems (ICDCS)*, pages 390–398.

562   Zhang, K., Mao, Y., Leng, S., He, Y., and ZHANG, Y. (2017). Mobile-edge computing for vehicular
563   networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology
564   Magazine*, 12(2):36–44.

565   Zhao, M. and Zhou, K. (2019). Selective offloading by exploiting arima-bp for energy optimization in
566   mobile edge computing networks. *Algorithms*, 12(2):48.

567   Zhao, Y. and Kim, B. (2020). Optimizing allocation and scheduling of connected vehicle service requests
568   in cloud/edge computing. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*,
569   pages 361–369.