electronics

MDPI

Article

A Fast and Efficient Task Offloading Approach in Edge-Cloud Collaboration Environment

Linyuan Liu [1,*], Haibin Zhu [2], Tianxing Wang [3] and Mingwei Tang [4]

[1] Department of E-Commerce, Nanjing Audit University, Nanjing 211815, China
[2] Collaborative Systems Laboratory, Nipissing University, North Bay, ON P1B 8L7, Canada; haibinz@nipissingu.ca
[3] School of Business, Nanjing Audit University, Nanjing 211815, China; wangtianxing@nau.edu.cn
[4] School of Computer Science, Nanjing Audit University, Nanjing 211815, China; tmw@nau.edu.cn
* Correspondence: liulinyuang@nau.edu.cn

Abstract: Edge-cloud collaboration fully utilizes the advantages of sufficient computing resources in cloud computing and the low latency of edge computing and better meets the needs of various Internet of Things (IoT) application scenarios. An important research challenge for edge-cloud collaboration is how to offload tasks to edge and cloud quickly and efficiently, taking into account different task characteristics, resource capabilities, and optimization objectives. To address the above challenge, we propose a fast and efficient task offloading approach in edge-cloud collaboration systems that can achieve a near-optimal solution with a low time overhead. First, it proposes an edge-cloud collaborative task offloading model that aims to minimize time delay and resource cost while ensuring the reliability requirements of the tasks. Then, it designs a novel Preprocessing-Based Task Offloading (PBTO) algorithm to quickly obtain a near-optimal solution to the Task Offloading problem in Edge-cloud Collaboration (TOEC) systems. Finally, we conducted extended simulation experiments to compare the proposed PBTO algorithm with the optimal method and two heuristic methods. The experimental results show that the total execution time of the proposed PBTO algorithm is reduced by 87.23%, while the total cost is increased by only 0.0004% compared to the optimal method. The two heuristics, although better than PBTO in terms of execution time, have much lower solution quality, e.g., their total costs are increased by 69.27% and 85.54%, respectively, compared to the optimal method.

Keywords: edge-cloud computing; task offloading; delay sensitivity; cost-efficient; reliability

check for updates

Citation: Liu, L.; Zhu, H.; Wang, T.; Tang, M. A Fast and Efficient Task Offloading Approach in Edge-Cloud Collaboration Environment. *Electronics* **2024**, *13*, 313. https://doi.org/10.3390/electronics13020313

Academic Editors: Fernando De la Prieta Pintado and Bahman Javadi

Received: 16 October 2023
Revised: 14 December 2023
Accepted: 8 January 2024
Published: 10 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the rapid development of wireless communication technology and the Internet of Things (IoT) technology, novel applications with high computing power and low latency requirements have emerged, such as real-time monitoring, smart manufacturing, autonomous driving, augmented reality, and online gaming [1–3]. One way to perform these applications is to offload them to the cloud. However, Cloud Servers (CSs) are usually far away from IoT devices, making it difficult to meet the latency requirements of these applications [4,5]. As an extension of cloud computing at the network edge, edge computing deploys computational resources on the Edge Servers (ESs) close to IoT devices [6]. Thus, offloading IoT tasks to ESs is beneficial to reduce service latency [7]. Unfortunately, due to resource constraints, it is difficult for ESs to fulfill large-scale task offloading requests.

Edge-cloud computing, as a new computing paradigm, combines the sufficient computing power of CSs and the low latency advantages of ESs, which can provide higher computing and transmission performance than single-edge computing or cloud computing [8,9]. Nevertheless, task offloading in Edge-cloud Collaborative Computing (ECC) is closely related to many factors, such as application goals, network environments, and

requirement constraints, and poses great challenges to its realization [10]. We summarize the following three major challenges.

First, time delay and resource cost are two important objectives for task offloading in ECC. While offloading tasks to ESs can reduce delay, ESs can be much more expensive to operate and maintain than CSs due to the lack of scale benefit [11]. On the contrary, CSs have cheaper resource prices but incur higher delays. Therefore, task offloading in ECC must consider the trade-off between time delay and monetary cost.

Second, in a dynamic, distributed, and open edge-cloud collaboration network, edge-cloud servers are prone to unpredictable failures, which can lead to interruptions in task processing and data loss [12,13]. To ensure the normal execution of tasks, a common remedy to improve task reliability is to use redundancy techniques, i.e., deploying multiple backup instances for each task in advance [14,15]. Although the backup technique improves the reliability of the tasks, it greatly increases the demand for computing resources, causing high resource costs. As a result, a task offloading model needs to be established for the ECC to ensure task reliability requirements while meeting resource capacity constraints.

Third, due to the heterogeneous nature of edge-cloud collaboration environments, each task requires a different dimension and number of resources, and each server has a different computing power and resource capacity [16]. Moreover, offloading tasks to different computing architectures for processing leads to significant differences in response time and resource cost [17]. Clearly, the heterogeneity of multi-task, multi-server, and multi-computing architecture scenarios increases the complexity of offloading decisions. Solving the combined time delay, resource cost, and reliability problem is very time-consuming. Especially when faced with large-scale task offloading requests, obtaining the optimal offloading solution within an acceptable delay is a very difficult task. Consequently, efficient and fast offloading algorithms need to be designed for the ECC.

Recently, many efforts have been devoted to the study of task offloading in ECC. In particular, the main objective of some of these works [18–26] is to reduce the task response time and device energy consumption. Additionally, many works [11,17,27] jointly consider the task offloading problem with delay, energy consumption, and resource cost objectives. However, these efforts do not consider the reliability requirements.

In reliability studies, some researchers [28–33] have considered the task offloading problem to have a trade-off between reliability and time delay, while others [34,35] have investigated the task offloading problem to have a minimum cost to ensure reliability. Most existing reliability studies [32,33] only require that the expected reliability of the system does not fall below a specified threshold. Since the failure probability of a task instance varies over time, it is difficult to accurately predict its reliability. Moreover, Li et al. [36] improved the overall reliability of the system by deploying multiple backup instances for tasks but did not require that each task instance be deployed. Therefore, the reliability of a specific task cannot be ensured.

The above research efforts generally focus on only one or a few optimization objectives. To the best of our knowledge, joint consideration of time delay, resource cost, and reliability for task offloading in ECC is still an open problem.

The task offloading in ECC is a complex combinatorial optimization problem. In recent years, several research works have employed heuristic methods [17,19,28,34,36], meta-heuristic methods [24–26], and machine learning methods [18,20,33] to solve the task offloading problem. Heuristic methods can generate solutions quickly but usually fall into local optima [37]. Meta-heuristics and machine learning methods usually produce high-quality solutions but tend to be time-consuming, and their runtimes are difficult to meet the demands of real-time IoT task offloading scenarios [38]. Moreover, to adapt to the dynamics of the edge-cloud collaboration environment, some machine learning algorithms need to collect new training samples to retrain the neural network, which reduces their performance [39]. Therefore, there is still much room for improvement of task offloading algorithms in edge-cloud collaboration systems.

E-CARGO is a powerful tool for complex collaborative systems modeling and is now widely used in industrial and social collaboration scenarios [40–42]. E-CARGO employs roles as the fundamental mechanisms for facilitating collaboration, describing collaborative systems and their key components in the form of a formal language. Based on this, researchers can use its six core concepts, including environment, class, agent, role, group, and object, to develop standard mathematical models related to the assignment problem and combinatorial optimization problem. The Group Multi-Role Assignment (GMRA) problem is a class of many-to-many assignment problems in E-CARGO [43,44]. In GMRA, each role needs to be assigned to multiple candidate agents to perform its job, and each agent has different qualifications for different roles and is able to take on more than one role at the same time. The optimization goal of GMRA is to form a group that has the largest overall qualification value by assigning agents to its roles.

An IoT system based on edge-cloud collaboration is a typical collaborative system that requires multiple different ESs and CSs to work together to provide processing for IoT tasks. Inspired by E-CARGO, we extend GMRA to describe the TOEC problem and explore new ideas for IoT task offload optimization. It is worth noting that the original GMRA only restricts the maximum number of roles an agent can be assigned. However, in the TOEC problem, to ensure the availability of each type of resource on the server, it is necessary to limit the number of specific types of resources required for all tasks assigned to the server. Therefore, the original GMRA lacks the ability to characterize the TOEC problem. In addition, the original GMRA problem is solved by an optimal method, but the optimal method makes it difficult to meet the delay-sensitive large-scale IoT task offloading requirements.

To address the above challenges, we propose a fast and efficient approach to solving the task offloading problem in ECC, which is inherently complex due to the dynamic nature of the ECC environment. The proposed approach effectively reduces time delay and resource cost, ensures task reliability, and achieves a near-optimal solution with low time overhead. Its novelties mainly include the following:

(1) In terms of modeling, it formally defines the TOEC problem model by extending GMRA, which takes into account various characteristics of the tasks, resource constraints of the servers, and heterogeneity of the edge-cloud collaboration environments, and its optimization objective is to minimize time delay and resource cost while satisfying reliability requirements and resource capacity constraints.
(2) In terms of algorithms, it designs a preprocessing-based task offloading algorithm that efficiently processes and offloads tasks while obtaining a near-optimal solution. These excellent features make it well-suited for delay-critical and large-scale IoT task offloading scenarios.

More specifically, our major contributions are summarized as follows:

(1) We formally define the task offloading problem in ECC by extending GMRA and formulating it as an Integer Linear Programming (ILP). Moreover, we evaluate the cost of the server to perform the task based on the delay cost and resource cost models.
(2) We design a novel preprocessing-based task offloading algorithm, which first reduces the size of the TOEC problem by selecting the Top k lower-cost candidate servers for each task and then solves the reduced TOEC problem using the IBM ILOG CPLEX optimization package (CPLEX) [45].
(3) We conducted a series of simulation experiments to compare the performance of our algorithm with the optimal method and two other baselines. The experimental results show that our algorithm effectively reduces the time delay and resource cost and achieves good performance of the near-optimal solution with low time overhead.

This article is organized as follows: it first reviews related work in Section 2. Then, Section 3 introduces a realistic scenario and formally defines the problem. Section 4 describes the cost evaluation method. Section 5 presents the design of the preprocessing

task offloading algorithm. Section 6 carries out simulation experiments to evaluate the performance of the proposed algorithm. Finally, the paper is concluded in Section 7.

2. Related Work

Recently, several efforts have been made in task offloading in cloud-edge collaborative computing. They typically consider delay, energy consumption, and the trade-off between delay and energy consumption as optimization objectives for task offloading.

Dai et al. [18] proposed a deep reinforcement learning-based task offloading scheme for vehicular edge computing in a cloud-edge environment, which minimizes the average processing latency of the tasks. Considering user delay satisfaction with services provided by Mobile Edge Computing (MEC), Li et al. [19] formulated the offloading problem with maximum cumulative satisfaction and presented an efficient heuristic algorithm to solve the problem. Tang et al. [20] proposed a dynamic resource allocation algorithm in a cloud-edge collaborative environment and obtained an optimal policy for task resource matching, which can effectively reduce network latency. Mazouzi et al. [21] formulated the task offloading problem as a nonlinear binary integer programming and minimized energy consumption with a distributed linear relaxation heuristic algorithm based on the Lagrangian decomposition method. Su et al. [22] proposed a cloud-edge collaborative computing offloading model for transmission energy and computation energy consumption and designed a near real-time computing offloading algorithm that can effectively reduce total energy consumption and achieve good performance close to the optimal solution.

In addition to considering a single delay or energy consumption objective, some studies have focused on time delay and energy consumption as tradeoffs. Liu et al. [23] proposed an online task offloading and resource allocation method for edge-cloud collaborative computing. The proposed method reduces the average latency and energy consumption of tasks and achieves near-optimal performance. Xu et al. [24] used the NSGA-III algorithm to solve the multi-objective optimization problem of shortening execution time and reducing the energy consumption for each mobile device. Laili et al. [25] proposed a large-scale task scheduling model in the edge-cloud collaboration environment, which is solved using a parallel swarm merge evolutionary algorithm and can effectively reduce system delay and energy consumption. Long et al. [26] proposed a task-offloading method based on the improved AR-MOEA algorithm for the task-offloading problem. Their work can reduce the total delay of task completion, energy consumption, and load difference among edge nodes.

To model heterogeneous application scenarios more realistically, several works have jointly considered the computation offloading problem with delay, energy, and cost objectives. Haber et al. [27] studied the optimal offloading problem with task energy and computation cost in a multi-tier edge architecture. Li et al. [11] investigated the task offloading and resource purchasing problem in an edge-cloud collaborative system and proposed a dual time scale Lyapunov optimization method, which purchases computing resources from the public cloud with different time scales and makes offloading decisions online. Hoseiny et al. [17] studied the task scheduling problem in a fog-cloud environment. They first formulated the problem as mixed-integer linear programming to minimize the computation, communication, and violation costs, and then they proposed two efficient heuristic algorithms that are able to obtain computationally inexpensive offloading schemes.

Fog or edge network-based IoT systems are prone to various failures as their distributed and open nature makes them vulnerable. These failures may occur during the operation of a server node, rendering it inoperable. In recent years, many research efforts have focused on task offloading problems that minimize time delay and energy consumption while enhancing reliability.

Wang et al. [28] studied the delay-sensitive and reliability-ensuring task offloading problem in fog computing and designed two algorithms, branch-and-bound and greedy heuristic, to solve the problem, which can realize shorter delay and lower energy consumption while ensuring reliability. Ghanavati et al. [29] proposed a task scheduling algorithm

based on dynamic fault-tolerant learning automata, which achieves efficient allocation of IoT tasks to fog nodes and can ensure reliable execution of tasks while optimizing response time and energy consumption. Hou et al. [30] investigated the fog computing-assisted task allocation problem for Unmanned Aerial Vehicle (UAV) swarms, aiming to minimize the energy consumption of UAV swarms under the constraints of latency and reliability, and developed a fast distributed algorithm based on the proximal Jacobi multiplier alternating direction method to solve the problem. Dong et al. [31] proposed an optimal computation offloading and resource allocation method for reliability-aware mobile edge computing, which can effectively reduce system latency and energy consumption. Liang et al. [32] studied the reliability-constrained task offloading problem in edge computing systems, aiming at minimizing latency with reliability as a constraint. The problem is addressed by a distributed reliability-aware task processing and offloading algorithm. Siyadatzadeh et al. [33] proposed a machine-learning-based primary-backup task assignment method in fog computing systems, which achieves enhanced system reliability while meeting real-time constraints by assigning the primary-backup task instances to appropriate fog nodes.

In addition to considering delay, energy consumption, and reliability objectives, some research efforts have also focused on task offloading for joint cost and reliability objectives. Yao et al. [34] investigated the trade-off between maximizing reliability and minimizing the system cost of fog resource allocation and designed a modified best-fit decreasing algorithm that was able to achieve suboptimal solutions with better time efficiency. Mao et al. [35] studied the computational offloading problem in mobile-edge computing, where the optimization objective is to minimize execution cost and task failures under energy threshold constraints, and developed a dynamic computational offloading algorithm based on Lyapunov optimization to solve the problem.

The above studies focusing on reliability generally require that the expected reliability of the system does not fall below a specified threshold. However, since the failure probability of a task instance varies over time, it is difficult to accurately predict its reliability. To this end, Li et al. [36] converted the user's reliability requirements into user satisfaction with the services provided by MEC network and improved the overall reliability of the system by deploying as many primary-backup instances as possible but did not require that the primary-backup instances must be deployed. As a result, the reliability of a specific task cannot be guaranteed.

Table 1 summarizes the related work and highlights the differences between this study and existing work. While researchers have done a great deal of useful work on task offloading for various optimization objectives, there is still much room for improvement in terms of system modeling and algorithmic efficiency.

As shown in Table 1, there are many previous works that have investigated task offloading models in fog or edge networks, which generally focus on one or a few optimization objectives, but there is still a lack of work on task offloading in ECC with joint consideration of time delay, resource cost, and reliability objectives. In our work, we propose an edge-cloud collaborative task offloading model that aims to minimize time delays and resource costs while ensuring reliability. Moreover, our work differs from related studies in terms of reliability-ensuring techniques. First, we improve reliability by adding the number of backup instances and do not need to predict the failure rate of task instances. Second, we require that each primary-backup instance must be assigned to a different server for execution, thus ensuring the reliability of each task.

In terms of task-offloading algorithms, several research efforts have used heuristic algorithms, meta-heuristic algorithms, and machine learning algorithms to solve the task-offloading problem. Heuristic algorithms can quickly obtain solutions, but the quality of the solutions is not high. Meta-heuristic algorithms (e.g., genetic algorithm, evolutionary algorithm, etc.) and machine learning algorithms (e.g., Q-learning algorithm, deep learning algorithm, etc.) can produce high-quality solutions, but they generally require long runtimes, posing challenges in meeting the real-time IoT task offloading requirements. Additionally, in edge-cloud collaboration scenarios, the number of tasks, the number of

servers, and the network conditions often change dynamically. As a result, some machine learning algorithms need to collect new training samples to retrain the neural network and adapt it to the new offloading environment, leading to a large limitation in their performance. Compared to previous studies, our proposed approach periodically collects and monitors relevant information about tasks and servers and invokes the offloading algorithm to solve an instance of the task offloading problem at a certain period, and thus can adapt to the dynamic changes of edge-cloud collaboration scenarios to a certain extent. The proposed preprocessing-based offloading algorithm demonstrates good performance in obtaining a near-optimal solution with low time overhead and is able to satisfy delay-sensitive large-scale IoT task offloading requirements.

Table 1. Comparison of Related Works.

References	System Model	Delay	Cost	Reliability	Task Dependency	Algorithm
[18]	Edge-cloud	√	×	×	×	Deep *Q*-network
[19]	Edge-cloud	√	×	×	×	Heuristic algorithm
[20]	Edge-cloud	√	×	×	√	Deep reinforcement learning
[21]	Edge-cloud	√	×	×	×	Linear relaxation heuristic
[23]	Edge-cloud	√	×	×	×	Online semi-decentralized algorithm
[24]	Edge-cloud	√	×	×	×	Genetic algorithm
[25]	Edge-cloud	√	×	×	√	Evolutionary algorithm
[26]	Edge-cloud	√	×	×	×	Evolutionary algorithms
[27]	Edge-cloud	√	√	×	×	Branch-and-Bound heuristic
[11]	Edge-cloud	√	√	×	×	Lyapunov optimization
[17]	Fog-cloud	√	√	×	×	Heuristic Algorithm
[28]	Fog	√	×	√	×	Branch-and-bound heuristic + Greedy heuristic
[29]	Fog	√	×	√	×	Learning automata
[30]	Fog	√	×	√	×	Alternatingdirection multiplier method
[31]	Edge	√	×	√	×	*N*-fold integer programming method
[32]	Edge	√	×	√	×	Alternatingdirection multiplier method
[33]	Fog	√	×	√	×	*Q*-learning
[34]	Fog	×	√	√	×	Best fit decreasing Algorithm
[35]	Edge	×	√	√	×	Lyapunov optimization
[36]	Edge	×	√	√	×	Heuristic Algorithm
This work	Edge-cloud	√	√	√	×	Preprocessing-based optimization algorithm

3. Problem Model

3.1. Real-World Scenario

To facilitate the upgrading of smart manufacturing levels, factory Y has deployed various IoT devices and machines in the workshops to collect real-time data and has developed a number of IoT applications to analyze the collected data for better condition monitoring and control. Alice and Bob are two important contributors to the IoT application scenario, where Alice is the Chief Executive Officer (CEO) of the factory, responsible for the planning and management of the IoT project, and Bob is the project director responsible for the specific implementation and execution of the IoT project. Alice plans to utilize the ECC architecture to provide task processing for the IoT applications. She appointed Bob to find an efficient task offloading method in ECC.

As shown in Figure 1, Bob designed a three-layer edge-cloud collaborative computing architecture containing a device layer, an edge layer, and a cloud layer. The device layer involves various IoT devices such as sensors, terminal devices, smart cameras, industrial robots, actuators, etc. The edge layer consists of a number of resource capacity-constrained computing devices, which are near the IoT devices and provide low-delay task processing

capability. In the cloud layer, there are powerful servers that are far away from IoT devices and have a high communication overhead.

The core component of the proposed three-layer computing architecture is the Edge-Cloud Broker (ECB), which is placed at the edge layer. The ECB consists of four subcomponents: task receiver, resource monitor, task scheduler, and management controller. The task receiver receives all task requests from IoT devices through the gateway. The resource monitor is responsible for collecting and monitoring the available resources of ESs and CSs periodically. The task scheduler runs offloading algorithms to solve the TOEC problem based on the characteristics and attributes of the tasks and servers. The management controller is responsible for managing and coordinating other components for task offloading.

In the computing architecture shown in Figure 1, it is assumed that the ECB and each ES have an associated Base Station (BS) deployed at the same location [22]. The ECB is wirelessly connected to the ESs through the associated BS and to the CSs through a high-speed fiber connection [17].

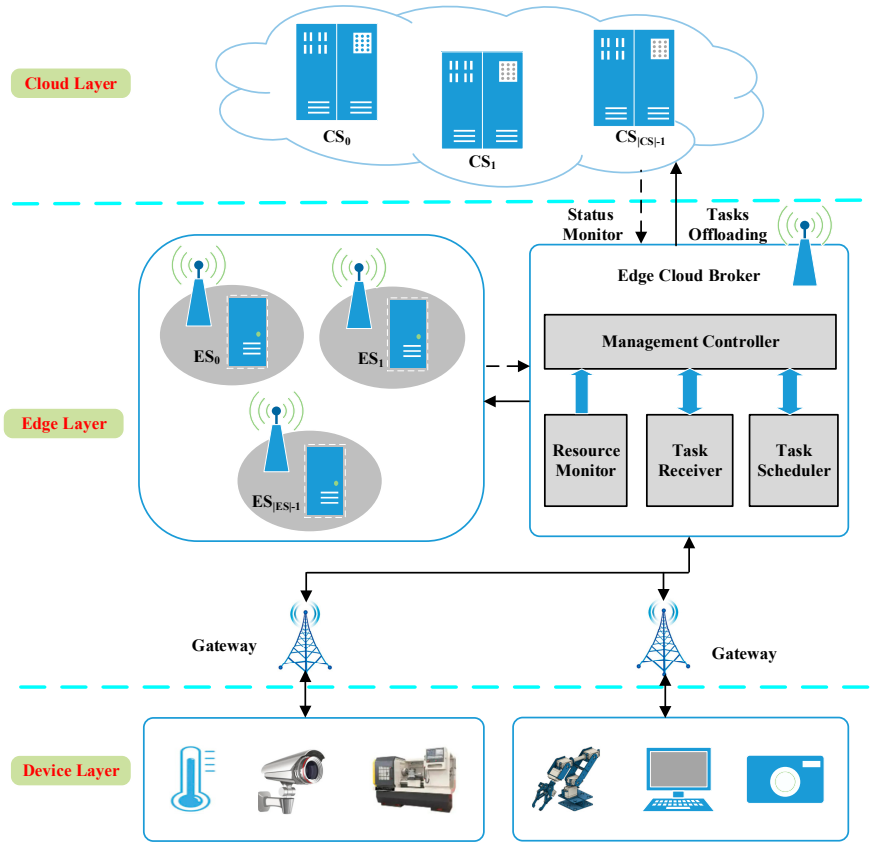


Figure 1. Three-layer edge-cloud collaborative computing architecture.

In Figure 1, the number and characteristics of the tasks and the number and resource capacity of the servers change dynamically and frequently. Therefore, ECB periodically collects and monitors these data and runs a task offloading algorithm based on them to solve an instance of the task offloading problem for each period. Finally, it offloads each task to the appropriate ES or CS for execution based on the offload solution. The detailed task offloading process is described below:

(1) IoT devices first submit their requests through the gateway to the task receiver, which analyzes the attributes of each submitted task, such as the size and type of the task, the deadline of the task, and the resource requirements of the task, and then sends them to the management controller, which then places them in a list of tasks to be assigned.

(2) When a specific period arrives, the management controller first obtains the available resource information of ESs and CSs from the resource monitor, then sets the reliability requirements based on experience as well as the characteristics and type of the tasks, and finally, it sends these data to the task scheduler.
(3) The task scheduler runs the offloading algorithm and forwards the offloading solution back to the management controller, which then assigns the tasks to the available ECs and CSs for execution.
(4) After the ES and CS have performed their tasks, they return the processing results to the controller, which checks whether the time delay of these results exceeds their deadlines. If not, it returns the results to the corresponding IoT devices.

To address the task offloading problem, Bob planned several reasonable requirements for achieving task offloading. First, to meet reliability requirements, each primary-backup instance of a task needs to be assigned to different servers. Second, when multiple tasks are assigned to the same server, the number of resources required by those tasks should not exceed the resource capacity of that server.

Bob realizes that he is experiencing a many-to-many assignment problem. Fortunately, he found E-CARGO and GMRA [40,43,46] to be effective ways to solve the assignment problem and decided to use them to solve the task offloading problem. However, the original GMRA problem only specifies the maximum number of tasks a server can take on and cannot directly deal with server resource capacity constraints. This is a new challenge for Bob. For this reason, we define the task offloading problem by extending GMRA and provide a solution for Bob in the following.

To better illustrate the task offload scenario, we provide an example including 10 tasks and 4 servers, where t_0–t_4 are small tasks with less resource demand, t_5–t_9 are large tasks with more resource demand, s_0, s_1 are ESs, and s_2, s_3 are CSs. The number of primary backup instances required by the tasks, the resource requirements and related attributes of the tasks, the resource capacity and unit price of the servers, and the cost incurred by the servers to perform the tasks are shown in Tables 2–7, respectively.

According to the reliability model in the literature [31], the reliability of a task is closely related to its length. This is because the longer the task length, the longer the execution time required on the server, which leads to an increased probability of failure. For large tasks with long task lengths, their reliability needs to be ensured more strongly, i.e., more task backups are required. Based on the above considerations, in Table 2, we set the number of primary-backup instances for small tasks (t_0–t_4) to 1–2, and the number of primary-backup instances for large tasks (t_5–t_9) to 2–3.

Based on Tables 3 and 4 and according to the delay cost model and resource cost model in Section 4, we can obtain the delay cost and resource cost required to offload tasks t_0–t_9 to servers s_0–s_3 for processing, as shown in Tables 5 and 6, respectively. We set the weight coefficient for balancing the importance of delay cost and resource cost to 0.5 and normalize the delay cost and resource cost in Tables 5 and 6 to obtain the cost incurred by servers s_0–s_4 to execute tasks t_0–t_9, as shown in Table 7. Based on the reliability requirements in Table 2 and the costs in Table 7, we call the task offloading algorithm in Section 5 to obtain the task offloading result, e.g., the total sum of the assigned evaluation values is 8.22, which is the minimum, and the optimal solution is shown in Table 7 (bold underlined). Note that the elements in Table 7 represent the costs incurred by the servers to perform the tasks, and the detailed evaluation process is described in Section 4.

Table 2. Primary-backup instances of the tasks.

Tasks	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
primary-backup instances	1	2	1	2	1	3	2	3	3	2

Table 3. Resource requirements and related attributes of the tasks.

Parameter	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
Computing requirements (MIPS)	449	248	260	307	675	1559	1108	1263	1346	1710
Memory requirements (GB)	0.30	0.11	0.49	0.32	0.48	0.98	0.94	0.57	0.51	0.96
Bandwidth requirements (Mbps)	16	9	13	10	18	39	45	39	21	40
Task length (MI)	500	700	800	900	1000	1500	2000	3000	3500	4000
Input data size (MB)	0.15	0.3	0.45	0.2	0.5	1	3.5	2	3.5	1.5

Table 4. Resource capacity and unit price of the servers.

Parameter	s_0	s_1	s_2	s_3
Computing capacity (MIPS)	2305	3209	8960	8089
Memory capacity (GB)	4.08	7.72	12.11	9.12
Bandwidth capacity (Mbps)	994	512	4391	4590
Computing usage price (G$/s)	0.6	0.9	0.3	0.45
Memory usage price (G$/MB)	0.003	0.004	0.001	0.002
Bandwidth usage price (G$/MB)	0.06	0.08	0.01	0.02

Table 5. The delay costs of offloading tasks to the servers for processing.

Servers	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
s_0	12.53	8.96	12.93	14.80	7.26	15.60	18.93	16.40	15.53	17.56
s_1	6.95	9.25	14.70	10.75	14.80	19.35	13.90	19.30	15.55	18.50
s_2	65.08	69.12	58.13	61.15	65.16	94.25	91.33	78.50	77.58	83.66
s_3	51.06	53.08	51.10	60.11	60.12	79.18	84.25	76.67	78.43	65.50

Table 6. The resource costs of offloading tasks to the servers for processing.

Servers	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
s_0	1.52	1.69	2.43	1.61	1.92	4.92	5.35	5.19	5.73	6.24
s_1	2.59	2.42	5.31	2.71	4.02	8.40	8.85	7.81	8.36	9.89
s_2	0.39	0.32	0.60	0.41	0.47	1.28	1.39	1.10	1.15	1.45
s_3	0.77	0.62	1.17	0.78	0.91	2.50	2.57	2.06	2.14	2.72

Table 7. The costs incurred by the servers to perform the tasks.

Servers	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
s_0	0.14	**0.13**	0.19	**0.16**	0.14	**0.33**	0.37	0.35	0.37	0.41
s_1	0.17	**0.17**	0.35	**0.19**	0.28	0.53	0.52	**0.50**	**0.51**	0.60
s_2	0.36	0.38	0.34	0.35	**0.37**	**0.56**	**0.55**	**0.47**	**0.47**	**0.52**
s_3	**0.31**	0.31	**0.33**	0.36	0.36	**0.55**	**0.58**	**0.51**	**0.52**	**0.48**

3.2. Problem Formulation

A typical task offloading scenario in ECC involves n tasks and m servers. All servers consist of a set of ESs and a set of CSs, denoted as $S = \{s_0, s_1, \ldots, s_{g-1}, s_g, s_{g+1}, \ldots, s_{m-1}\}$, where $S^E = \{s_0, s_1, \ldots, s_{g-1}\}$ is a set of ESs, and $S^C = \{s_g, s_{g+1}, \ldots, s_{m-1}\}$ is a set of CSs, $S = S^E \cup S^C$. It is assumed that, in each period, each IoT device's offload request contains only one indivisible task [9,38]. All tasks to be offloaded are denoted as $T = \{t_0, t_1, \ldots, t_{n-1}\}$.

For each task $t_j \in T$ $(0 \leq j < n)$, it is defined as an eight-tuple $t_j = <in_j, out_j, lr_j, cr_j, mr_j, br_j, rr_j, dl_j>$, where in_j and out_j denote t_j's data sizes of the input and output, respectively; lr_j denotes the length of t_j (in terms of Million Instructions-MI); cr_j, mr_j, and br_j denote the number of computational, memory, and bandwidth resources required by t_j, respectively; rr_j denotes the reliability requirement of t_j, which is the number of backup instances

required by t_j; dl_j denotes the deadline for t_j, which specifies the maximum tolerable delay. The resource capacity and price of each server $s_i \in S$ ($0 \leq i < m$) is characterized by the six-tuple $s_i = <cc_i, mc_i, bc_i, cp_i, mp_i, bp_i>$, where cc_i, mc_i, and bc_i denote the CPU computational power, memory capacity, and bandwidth capacity provided by s_i, respectively; and cp_i, mp_i, and bp_i denote unit price of the computational, memory, and bandwidth resources of the server s_i, respectively. The main notations used in this paper are given in Table 8.

Table 8. Symbol Notations.

Notation	Description
T	Set of tasks, where $\|T\| = n$
S	Set of servers, where $\|S\| = m$
S^E	Set of edge servers, where $\|S^E\| = g$
S^C	Set of cloud servers, where $\|S^C\| = m - g$
in_j, out_j	The input (output) data size of task t_j
lr_j	The length of task t_j
rr_j	The reliability requirement of task t_j
cr_j, mr_j, br_j	The computing, memory, and bandwidth resources requirement by task t_j
cc_i, mc_i, bc_i	The capacity of server s_i in terms of computing, memory, and bandwidth resources
cp_i, mp_i, bp_i	The unit price of the computing, memory, and bandwidth resources of server s_i
L	The role range vector
Q	The cost matrix
R^h	The resource requirement vector, and $R^h[j]$ indicates the hth-type resource required by t_j
W^h	The resource capacity vector, and $W^h[i]$ indicates the hth-type resource capacity provided by s_i
P	The assignment matrix, and $P[i, j]$ indicates whether server i is assigned to task j
σ	The total cost of the TOEC problem
$\tau_{b,i}^{tran}$, $\tau_{j,i}^{exe}$, $\tau_{i,j}$	The transmission delay, execution delay, and total delay cost for task j to be offloaded to server i for execution
$\varsigma_{i,j}^{cpu}$, $\varsigma_{i,j}^{mem}$, $\varsigma_{i,j}^{bw}$, $\varsigma_{i,j}$	The computing cost, memory cost, bandwidth cost, and total resource cost of executing task j on server i
α	The weighting coefficient for delay cost and resource cost objectives
Q_{C_j}	The column vector of the elements at column j in Q
$\min^k Q_{C_j}$	The ordered vector of the k smallest elements picked from Q_{C_j}
$V_{C_j}^k$	The row index vector that records the corresponding row index number in Q for each element in $\min^k Q_{C_j}$
Q_{R_i}	The row vector of the elements at row i in Q
V_{R_i}	The column index vector that keeps an ordered record of the corresponding column index number in Q of each element remaining in Q_{R_i}
$\overline{\sigma}$	The total cost of the RTOEC problem

The TOEC problem is really about how to assign multiple tasks to multiple servers to minimize delay costs and resource costs while ensuring reliability. Inspired by the theory of E-CARGO and GMRA, we extend GMRA to give a formal definition of the TOEC problem, where tasks are roles and servers are agents.

Definition 1 ([41,43]). *A role range vector L is a vector representing the lower bound of the role ranges in the environment e of group g.*

In our scenario, to enhance reliability, multiple instances of each task are created: a primary task instance and multiple backup task instances, where the number of backup instances indicates the reliability requirements of the task. L reflects the number of primary-backup instances of the tasks, i.e., $L[j] = 1 + rr_j$ ($0 \leq j < n$); if no backup is required, $L[j] = 1$, and if a backup is required, $L[j] = 2$. Both the primary and backup task instances need to be offloaded to different servers for execution. For example, each task in Table 2 specifies the

number of primary-backup instances; hence, the L vector based on the tasks in Table 2 is L = [1, 2, 1, 2, 1, 3, 2, 3, 3, 2].

Note that at each task offloading period, we have difficulty in accurately predicting the failure rates. Therefore, it is not easy to set $L[j]$ correctly, and we may need to conduct an in-depth study on this topic in the future. Here, we propose a preliminary scheme for setting $L[j]$: initially, ECB estimates the average failure rate of ECs and CSs based on experience; then, ECB integrates the average failure rate of the servers, the length of task t_j, and its criticality to determine $L[j]$ accordingly.

Definition 2. *A cost matrix Q is an $m \times n$ matrix, where $Q[i, j] \in [0, 1]$ expresses the cost of agent i ($0 \leq i < m$) to take on role j ($0 \leq j < n$), with $Q[i, j] = 0$ meaning the lowest and 1 the highest.*

In the scenario described in Section 3, the Q matrix is evaluated based on the delay cost and resource cost incurred by the server in providing processing to the task. Building a Q matrix for the TOEC problem is not trivial and requires calculating multiple types of time delays and multiple dimension resource costs. Accordingly, we present a more comprehensive evaluation method to establish Q in Section 4. Table 5 illustrates the normalized Q matrix for the example in Section 3.

To describe the resource requirements of the task and the capacity limits of the server, we formalize the following two data structures.

Definition 3. *A resource requirement vector R^h is an n-vector, where $R^h[j]$ ($0 \leq j < n$) indicates the number of the hth type of resource required by task t_j.*

Note that in this paper, we mainly consider three types of resources: computational, memory, and bandwidth, i.e., h = 0, 1, or 2 denotes computational, memory, and bandwidth resources, respectively, i.e., for a task t_j ($0 \leq j < n$), there are $R^0[j] = cr_j$, $R^1[j] = mr_j$, and $R^2[j] = br_j$.

Definition 4. *A resource capacity vector W^h is an m-vector, where $W^h[i]$ ($0 \leq i < m$) indicates the capacity of the hth type of resource provided by server s_i.*

Similar to the resource requirement vector, the resource capacity vector also involves three main types of resources, namely computational, memory, and bandwidth, i.e., for a server s_i ($0 \leq i < m$), there are $W^0[i] = cc_i$, $W^1[i] = mc_i$, and $W^2[i] = bc_i$.

Definition 5 ([41,43])**.** *An assignment matrix P is defined as an $m \times n$ matrix, where $P[i, j] \in \{0, 1\}$ ($0 \leq i < m$, $0 \leq j < n$) indicates whether or not agent i is assigned to role j, with $P[i, j] = 1$ meaning yes and 0 meaning no.*

The assignment matrix represents an offloading solution in our scenario, where $P[i, j] \in \{0, 1\}$ expresses an offloading decision, with $P[i, j] = 1$ meaning to offload task j to server i and 0 meaning no. For example, the task assignment matrix for the example in Section 3 is shown in Table 9.

Table 9. Assignment matrix P.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
s_0	0	1	0	1	0	1	0	0	0	0
s_1	0	1	0	1	0	0	0	1	1	0
s_2	0	0	0	0	1	1	1	1	1	1
s_3	1	0	1	0	0	1	1	1	1	1

Definition 6 ([41,43])**.** *Role j is workable in the group if it has been assigned enough agents, that is,*

$$\sum_{i=0}^{m-1} P[i,j] \geq L[j]. \quad (1)$$

Definition 7 ([41,43]). *P is workable if each role in group g is workable, i.e.,*

$$\sum_{i=0}^{m-1} P[i,j] \geq L[j]. \quad (2)$$

Only if P is workable, the group g is workable.

Definition 8. *The total cost is defined as the sum of the assigned agents' costs, that is,*

$$\sigma = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} Q[i,j] \times P[i,j]. \quad (3)$$

Definition 9. *Given L, Q, R^h, and W^h, the TOEC problem is to find a workable P to*

$$\text{Min}\sigma = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} Q[i,j] \times P[i,j],$$

subjected to

$$P[i,j] \in \{0,1\} \qquad (0 \leq i < m,\ 0 \leq j < n), \quad (4)$$

$$\sum_{i=0}^{m-1} P[i,j] = L[j] \qquad (0 \leq j < n), \quad (5)$$

$$\sum_{j=0}^{n-1} R^h[j] \times P[i,j] \leq W^h[i] \qquad (0 \leq i < m,\ 0 \leq h < 3), \quad (6)$$

where

(1) *Expression (4) is the binary value indicating whether server i is assigned to task j or not;*
(2) *Expression (5) indicates that each task must be assigned to enough servers to ensure the reliability requirements of the task;*
(3) *Expression (6) depicts that the hth resource required by all tasks assigned to s_i cannot exceed the capacity of its hth resource.*

The TOEC problem defined in Definition 9 can be formulated as an ILP problem, which we can transform into the standard form of ILP using the following steps:

(1) The ILP problem [47] is a class of problems to find a vector $X = [x_0, x_1, \ldots, x_{n^*-1}]$ to obtain

$$\text{Min (Max)} \sum_{j^*=0}^{n^*-1} c_{j^*} x_{j^*}, \quad (7)$$

subjected to

$$x_{j^*} \in \{0,1\} \qquad (0 \leq j^* < n^*) \quad (8)$$

$$\sum_{j^*=0}^{n^*-1} a_{i^*,j^*} x_{j^*} \leq (=,\ \geq)\ b_{i^*} \qquad (0 \leq i^* < m^*). \quad (9)$$

(2) From Definition 9, we have L, Q, R^h, W^h, and P.

Let $n^* = m \times n$, $c_{j^*} = Q[j^*/n, j^*\%n]$ $(0 \leq j^* < n^*)$, $x_{j^*} = P[j^*/n, j^*\%n]$ $(0 \leq j^* < n^*)$.

We have $\sum_{j^*=0}^{n^*-1} c_{j^*} x_{j^*} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} Q[i,j] \times P[i,j]$. That is, the objective of the ILP and that of the TOEC problem are the same. At the same time, Expression (4), $P[i,j] \in \{0,1\}$ $(0 \leq i < m, 0 \leq j < n)$, is the same as Expression (8), i.e., $x_{j^*} \in \{0,1\}$ $(0 \leq j^* < n^*)$.

(3) Let

$m^* = n + m + m + m$
$b_{i^*} = L[j]$ $(0 \leq j < n, 0 \leq i^* < n, i^* = j)$
$b_{i^*} = W^0[i]$ $(n \leq i^* < n + m, 0 \leq i < m, i^* = i + n)$
$b_{i^*} = W^1[i]$ $(n + m \leq i^* < n + m + m, 0 \leq i < m, i^* = i + n + m)$
$b_{i^*} = W^2[i]$ $(n + m + m \leq i^* < n + m + m + m, 0 \leq i < m, i^* = i + n + m)$

$a_{i^*,i^*+l\times n} = 1$ $(0 \le i^* < n, 0 \le l < m)$
$a_{i^*,l\times n+j^*\%n} = R^0[j]$ $(n \le i^* < n+m, 0 \le j^* < n^*, 0 \le l < n, 0 \le j < n)$
$a_{i^*,l\times n+j^*\%n} = R^1[j]$ $(n+m \le i^* < n+m+m, 0 \le j^* < n^*, 0 \le l < n, 0 \le j < n)$
$a_{i^*,l\times n+j^*\%n} = R^2[j]$ $(n+m+m \le i^* < n+m+m+m, 0 \le j^* < n^*, 0 \le l < n, 0 \le j < n)$
Then, Expressions (5) and (6) are the same as Expression (9), i.e.,
$\sum_{j^*=0}^{n^*-1} a_{i^*,j^*} x_{j^*} = b_{i^*} (0 \le i^* < n)$, and $\sum_{j^*=0}^{n^*-1} a_{i^*,j^*} x_{j^*} \le b_{i^*} (n \le i^* < m^*)$.
In summary, the TOEC problem is expressed as an ILP problem.

4. Cost Evaluation

The optimization objective of the TOEC problem is to minimize the delay cost and resource cost. The delay cost model and the resource cost model are given below.

4.1. Delay Cost Model

In the ECC architecture shown in Figure 1, we mainly consider two types of time delay: (1) transmission delay and (2) execution delay. Their calculation processes are described in detail below.

(1) Transmission delay

When task t_j is offloaded to the ES_i for processing, the ESB utilizes the wireless network to transmit the data of t_j to the BS near the ES_i, which in turn passes it to the ES_i for processing. The transmission delay includes the time incurred by transmitting data from the ECB to the ES_i, the data delivering time between the BS and the ES_i, and the time incurred by returning the processing result to the ECB after t_j is executed on the ES_i. Considering that each ES and the associated BS are deployed at the same location, the data transfer between them is ignored. Moreover, we ignore the result return time since the size of data in the processing result is usually small [19].

Similarly, when task t_j is offloaded to the CS_i for processing, the ESB transmits the data of t_j to the CS_i using the wired network. The transmission delay includes the time incurred by transferring the data from the ECB to the CS_i, as well as the time incurred by the CS_i to return the processing result to the ECB after executing t_j. We still assume that the return time of the processing result is ignored.

According to Shannon's theorem [48], the data transmission rate of the channel between the ECB and the ES s_i can be expressed as follows:

$$r_{b,i} = \omega \log_2(1 + SNR_{b,i}) \tag{10}$$

where b indicates the BS co-located with the ESB, ω is the bandwidth of the wireless channel, and $SNR_{b,i}$ is the wireless channel's signal-to-noise ratio.

Based on Equation (10), the data transmission delay between the ECB and the server s_i for task t_j can be defined as follow:

$$\tau_{b,i}^{tran} = \begin{cases} \frac{in_j}{r_{b,i}} & \text{if } s_i \in S^E \\ \frac{in_j}{\varpi_{b,i}} & \text{else if } s_i \in S^C \end{cases}, \tag{11}$$

where $\varpi_{b,i}$ is the bandwidth of the wired link between the BS b and the CS s_i.

(2) Execution delay:

The delay for executing the task t_j on a specific server (ES or CS) s_i can be calculated using

$$\tau_{j,i}^{exe} = \frac{lr_j}{cc_i}. \tag{12}$$

In summary, the delay cost for task t_j to be offloaded to server s_i for execution can be expressed as

$$\tau_{i,j} = \tau_{b,i}^{tran} + \tau_{j,i}^{exec}. \tag{13}$$

4.2. Resource Cost Model

In an ECC system, offloading tasks to servers for processing consumes multiple resources and incurs various costs, including computational costs, memory costs, and bandwidth usage costs.

The computational cost is the product of the price of CPU processing per unit time of server i and the execution time spent by task j on server i, which is calculated using

$$\varsigma_{i,j}^{cpu} = cp_i \times \tau_{j,i}^{exec}. \tag{14}$$

The memory cost is the product between the price per storage unit of server i and the number of memory units required by task j, which is expressed as

$$\varsigma_{i,j}^{mem} = mp_i \times mr_j. \tag{15}$$

The bandwidth cost is the product of the price per data unit of server i and the sum of the input and output data sizes for task j, which is defined as follow:

$$\varsigma_{i,j}^{bw} = bp_{\,i} \times (in_j + out_j). \tag{16}$$

According to Equations (14)–(16), the resource cost of executing task j on server i can be calculated using

$$\varsigma_{i,j} = \varsigma_{i,j}^{cpu} + \varsigma_{i,j}^{mem} + \varsigma_{i,j}^{bw}. \tag{17}$$

Based on the above delay cost and resource cost models, we evaluate the cost of the server to perform the task in the following. Specifically, each $Q[i, j]$ is obtained by evaluating the weighted sum of the normalized delay cost and resource cost incurred by server i to execute task j, that is,

$$Q[i, j] = \alpha \times \tau_{i,j}^{*} + (1 - \alpha) \times \varsigma_{i,j}^{*}, \tag{18}$$

where $\alpha \in [0, 1]$ is a weighting coefficient to balance the importance of delay cost and resource cost, e.g., when $\alpha > 0.5$, it indicates that delay cost is more important than resource cost. $\tau_{i,j}^{*}$ and $\varsigma_{i,j}^{*}$ denote the normalized delay cost and resource cost, respectively, which can be calculated using

$$\tau_{i,j}^{*} = \frac{\tau_{i,j}}{\tau^{\max}} \tag{19}$$

$$\varsigma_{i,j}^{*} = \frac{\varsigma_{i,j}}{\varsigma^{\max}}, \tag{20}$$

where $\tau^{\max}$ and $\varsigma^{\max}$ denote the maximum possible delay cost and resource cost of task t_j, respectively.

In the scenario outlined in Section 3, the lengths, input file sizes, and the number of computational, memory, and bandwidth resources required by tasks t_0–t_9 are shown in Table 3, and the servers' unit price for computational, memory, and bandwidth resources is shown in Table 4. We assume that the transmission delays for the tasks offloaded to the ESs and CSs randomly vary within the ranges of 5–15 ms and 50–150 ms, respectively. Based on the delay cost and resource cost models, Tables 5 and 6 show the delay cost and resource cost of offloading tasks to the servers for processing, respectively.

Based on Tables 5 and 6 and setting the weight coefficient α to 0.5, the costs incurred by the servers s_0–s_4 to perform the tasks t_0–t_9 can be evaluated according to Equations (18)–(20), as shown in Table 7.

5. Solution to the TOEC Problem

In Section 3, the TOEC problem is formulated as an ILP. To solve this problem, it is actually necessary to assign n tasks to m servers. Since the requirements of the tasks and the properties of the servers are different, m^n solutions exist for assigning tasks to the servers.

In a large-scale scenario, n and m are usually on the scale of hundreds or thousands [17]. Meanwhile, to meet the low delay requirement of IoT applications, offloading decisions need to be made immediately once the requests arrive [49]. Therefore, designing an efficient offloading algorithm is a challenging task.

Fortunately, in the experimental evaluation, we observe that the optimal assignment results for each task are generally distributed among the few lowest-cost candidate servers despite the constraints on server resource capacity. Inspired by this phenomenon, we propose the PBTO algorithm, which is based on the following ideas:

(1) The candidate servers for each task are sorted according to their cost based on the Q matrix.
(2) The Top k candidate servers are picked out for each task to reduce the size of the TOEC problem, and finally, CPLEX is invoked to solve the Reduced TOEC (RTOEC) problem.

To facilitate the algorithm description, the following auxiliary data structures are defined.

Definition 10. *A column vector Q_{C_j} ($0 \le j < n$) is the vector of the elements at column j in Q, where $Q_{C_j}[i]$ ($0 \le j < n$, $0 \le i < m$) denotes the ith element in Q_{C_j}, which is located in the ith row and jth column of Q, $|Q_{C_j}| = m$.*

Definition 11. *Let $\min^k Q_{C_j}$ be an ordered vector of the k smallest elements picked from Q_{C_j}, where $|\min^k Q_{C_j}| = k$, and $k \le m$. The row index vector $V^k_{C_j}$ is used to record the corresponding row index number in Q for each element in $\min^k Q_{C_j}$, where $V^k_{C_j}[i'] (0 \le i' < k)$ expresses the ith element of $V^k_{C_j}$, and for any $Q_{C_j}[i] \in Q_{C_j}$ ($0 \le i < m$), if there exists $Q_{C_j}[i] \in \min^k Q_{C_j}$, then $V^k_{C_j}[i'] = i$.*

Definition 12. *A row vector Q_{R_i} ($0 \le i < m$) is the vector of the elements at row i in Q, where $Q_{R_i}[j]$ ($0 \le j < n$, $0 \le i < m$) denotes the jth element of Q_{R_i}, which is located in the ith row and jth column of Q, $|Q_{R_i}| = n$.*

Definition 13. *The column index vector V_{R_i} is used to keep an ordered record of the corresponding column index number in Q of each element remaining after Q_{R_i} has been reduced, where $V_{R_i}[j']$ ($0 \le j' < n$) expresses the jth element of V_{R_i}, $|V_{R_i}| \le n$, and for any $Q_{R_i}[j] \in Q_{R_i}$ ($0 \le j < n$), if there exists $i \in V^k_{C_j}$, then $V_{R_i}[j'] = j$.*

Definition 14. *Based on $V^k_{C_j}$ and V_{R_i}, the total cost of the RTOEC problem is defined as*

$$\overline{\sigma} = \sum_{i'=0}^{|V^k_{C_j}|-1} \sum_{j=0}^{n-1} Q[V^k_{C_j}[i'], j] \times P[V^k_{C_j}[i'], j]. \tag{21}$$

Definition 15. *Given L, Q, R^h, C^h, $V^k_{C_j}$, and V_{R_i}, the RTOEC problem is to find a workable P to*

$$\mathrm{Min}\overline{\sigma} = \sum_{i'=0}^{|V^k_{C_j}|-1} \sum_{j=0}^{n-1} Q[V^k_{C_j}[i'], j] \times P[V^k_{C_j}[i'], j],$$

subjected to:

$$P[V^k_{C_j}[i'], j] \in \{0, 1\} \quad (0 \le i' < \left|V^k_{C_j}\right|, \ 0 \le j < n), \tag{22}$$

$$\sum_{i'=0}^{|V^k_{C_j}|-1} P[V^k_{C_j}[i'], j] = L[j] \quad (0 \le j < n), \tag{23}$$

$$\sum_{j'=0}^{|V_{R_i}|-1} R^h[V_{R_i}[j']] \times P[i, V_{R_i}[j']] \le W^h[i] \quad (0 \le i < m, \ 0 \le h < 3). \tag{24}$$

The PBTO algorithm consists of two phases: the preprocessing phase and the task offloading phase. The specific process of the preprocessing phase is shown in Algorithm 1, and its main steps are as follow:

Step 1: First, copy the elements of the jth column of Q into Q_{C_j}, sort the elements of Q_{C_j} in ascending order, and pick out the Top k elements to be added to $\min^k Q_{C_j}$; then obtain the corresponding row index number of each element in $\min^k Q_{C_j}$, and save them one by one into the row index vector $V_{C_j}^k$; and finally, add $V_{C_j}^k$ into the set of row index vectors RS.

Step 2: First, copy the ith column element of Q into Q_{R_i}; then, traverse each element $Q_{R_i}[j]$, and determine whether its row index number exists in row index vector $V_{C_j}^k$; if the result is true, obtain the corresponding column index number of $Q_{R_i}[j]$ in Q, and save it into column index vector V_{R_i}; and finally, add V_{R_i} into the set of column index vectors CS.

Algorithm 1: Preprocessing Algorithm

Input:
T: the tasks set; S: the servers set; Q: the cost matrix.
Output:
RS, CS: the row index vector set, and the column index vector set.

```
for each t_j ∈ T do
    Copy all elements in column j of Q into Q_{C_j};
    Sort the elements of Q_{C_j} in ascending order;
    min^k Q_{C_j} ← Pick the first k elements from Q_{C_j};
    Save the corresponding row index number in
    Q for each element in min^k Q_{C_j}
    into V_{C_j}^k;
end for
Add V_{C_j}^k into RS;
for each s_i ∈ S do
        Copy all elements in row i of Q into Q_{R_i};
        for each Q_{R_i}[j] ∈ Q_{R_i} do
            if Q_{R_i}[j]'s row index number i satisfies i ∈ V_{C_j}^k then
            Save the corresponding column index number in Q of the element
                Q_{R_i}[j] into V_{R_i};
            end if
        end for
end for
Add V_{R_i} into CS;
return RS, CS
```

From the inputs of Algorithm 1, it can be observed that the number of tasks, the number of servers, and the cost matrix Q are key parameters that affect the efficiency of the PBTO algorithm. Due to the dynamic and open nature of edge-cloud collaboration systems, these parameters, including the number of tasks, the number of servers, the characteristics and resource requirements of tasks, and the available resources of servers, change dynamically. Therefore, the task offloading scheme in ECC should be adjusted dynamically according to their changes. To address this challenge, we present our task offloading scheme in Section 3. In this scheme, the ECB periodically collects and monitors these parameters, and when each offloading period arrives, the ECB first updates the above parameters based on the latest data collected and then invokes the PBTO algorithm to solve an instance of the offloading problem based on the updated parameters.

In addition, in Section 4, the Q matrix is created by evaluating the delay cost and resource cost incurred by tasks executed on the server, which in turn depend on network conditions. In this paper, we do not currently consider the impact of dynamically changing

network conditions on delay cost and resource cost, and we plan to investigate this issue in-depth in future work.

The task offloading phase solves the RTOEC problem by invoking CPLEX based on the preprocessing results of Algorithm 1. The detailed process is shown in Algorithm 2, and its main steps are as follows:

Step 1: The matrices Q and P were converted into the corresponding one-dimensional matrices U and X.

Step 2: Based on U and X, first, an objective expression is declared; then, each task t_j in the task set T is traversed, and the row index vector $V_{C_j}^k$ corresponding to t_j is obtained; then, each element $V_{C_j}^k[i']$ in $V_{C_j}^k$ is traversed to find its corresponding element in U and X and added into the objective expression; finally, the minimization optimization objective is established.

Step 3: Based on X and L, for each task t_j in the task set T, a reliability constraint expression is declared, and its row index vector $V_{C_j}^k$ is obtained; then, each of its elements $V_{C_j}^k[i']$ is traversed, and the corresponding element of $V_{C_j}^k[i']$ in X is found and added into the reliability constraint expression; finally, the reliability constraint equation is established.

Step 4: Based on X, R^0, R^1, R^2, W^0, W^1, and W^2, for each server s_i in the server set S, firstly, the three constraint expressions for computing, memory, and bandwidth capacity are declared, and the column index vector V_{R_i} of s_i is obtained; then, each element $V_{R_i}[j']$ in V_{R_i} is traversed to find its corresponding element in X, the total amount of demand for each type of resources of the element is obtained and added to the corresponding resource constraint expression, and finally, the three constraint inequalities for computing, memory, and bandwidth are established.

Step 5: Based on the above objective and constraints, cplex.solve() method is invoked to obtain the optimal solution.

The complexity of the above algorithms depends on the number of tasks (n) and the number of servers (m). The time complexity of Algorithm 1 is decided using the following: (1) the time complexity of step 1 is $O(n \times m\log m)$, and (2) the time complexity of step 2 is $O(m \times n)$.

Notably, Algorithm 1 preprocesses the TOEC problem, by which only the Top k candidate servers are selected for each task, thus effectively reducing the number of decision variables. For example, the task offloading problem for n tasks and m servers has $m \times n$ decision variables, while in PBTO, the number of decision variables is reduced to $n \times k$.

The time complexity of the first four steps of Algorithm 2 is determined using the following: (1) the time complexity of step 1 is $O(n \times m)$, (2) the time complexity of step 2 is $O(n \times k)$, (3) the time complexity of step 3 is $O(n \times k)$, and (4) the time complexity of step 4 is $O(n \times m)$. Step 5 of Algorithm 2 employs CPLEX (version 22.1.0) to solve the RTOEC problem. CPLEX is a commercial optimization software package supported by IBM. We do not know the actual complexity of CPLEX. The simplex, ellipsoid, and interior point algorithms are commonly used for solving linear programming, and their worst-case time complexities are 2^{Φ}, $\Phi^4 \times \Omega$, and $\Phi^{3.5} \times \Omega$, respectively, where Φ is the number of decision variables, and Ω is the length of a binary coding of the input data, that is [50]

$$\Omega = \sum_{i=0}^{m} \sum_{j=0}^{n} \lceil \log_2(|a_{i,j}|+1) + 1 \rceil, \tag{25}$$

where $|a_{i,j}|$ denotes the number of binary bits of element $a_{i,j}$ in the input data matrix A.

Algorithm 2: Task Offloading Algorithm

Input:
Q: the cost matrix; L: the role range vector;
R^0, R^1, R^2: the computing, memory, bandwidth resources requirement vector;
W^0, W^1, W^2: the computing, memory, bandwidth resources capacity vector;
RS, CS: the row index vector set, and the column index vector set.
Output:
P: the task assignment matrix.

Convert the Q matrix into a one-dimensional matrix U;
Convert the P matrix into a one-dimensional matrix X;
IloLinearNumExpr *obj* = cplex.linearNumExpr();
for each $t_j \in T$ **do**
  $V_{C_j}^k \leftarrow$ Get the row index vector of column j from RS;
  for each element $i' \in V_{C_j}^k$ **do**
    obj.addTerm$\Big(U[j + \ \times V_{C_j}^k[i']\,],\ X[j + \ \times V_{C_j}^k[i']\Big)$;
  end for
end for
cplex.addMinimize(*obj*);
for each $t_j \in T$ **do**
  IloLinearNumExpr *cons_r* = cplex.linearNumExpr();
$V_{C_j}^k \leftarrow$ Get the row index vector of column j from RS;
  for each element $i' \in V_{C_j}^k$ **do**
    cons_r.addTerm$\Big(1,\ X[j + \ \times V_{C_j}^k[i']\,\Big)$;
  end for
  cplex.addEq(*cons_r*, $L[j]$);
end for
for each $s_i \in S$ **do**
  IloLinearNumExpr *cons_c* = cplex.linearNumExpr();
  IloLinearNumExpr *cons_m* = cplex.linearNumExpr();
  IloLinearNumExpr *cons_b* = cplex.linearNumExpr();
$V_{R_i} \leftarrow$ Get the column index vector of row i from CS;
  for each element $j' \in V_{R_i}$ **do**
    cons_c.addTerm($R^0[V_{R_i}[j']],\ X[V_{R_i}[j'] + i \times n]$);
    cons_s.addTerm($R^1[V_{R_i}[j']],\ X[V_{R_i}[j'] + i \times n]$);
    cons_b.addTerm($R^2[V_{R_i}[j']],\ X[V_{R_i}[j'] + i \times n]$);
  end for
  cplex.addLe(*cons_c*, $W^0[i]$);
  cplex.addLe(*cons_m*, $W^1[i]$);
  cplex.addLe(*cons_b*, $W^2[i]$);
end for
$P \leftarrow$ invoke cplex.solve () to get the optimal assignment solution;

We assume that CPLEX integrates simplex, ellipsoid, and interior point algorithms to solve linear programming problems. The input data for Algorithm 2 consist of Q, R^0, R^1, R^2, W^0, W^1, W^2, and L, where Q is an $n \times m$ double-type matrix; R^0, R^1, and R^2 are n-dimensional double-type vectors; W^0, W^1, and W^2 are m-dimensional double type vectors; L is an n-dimensional integer-type vector. Generally, an integer variable needs to occupy 32 bits, and a double variable needs to occupy 64 bits. Moreover, since PDTO preprocesses the TOEC problem, the elements in Q are correspondingly reduced to $n \times k$ double-type variables. In summary, the number of bits required for the input data of Algorithm 2 is

$$\begin{aligned}
&\textstyle\sum_{i=0}^{k-1}\sum_{j=0}^{n-1}\lceil\log_2(|Q[i,j]|+1)+1\rceil+3\times\sum_{j=0}^{n-1}\left\lceil\log_2(\left|R^h[j]\right|+1)+1\right\rceil+\\
&\textstyle 3\times\sum_{i=0}^{m-1}\left\lceil\log_2(\left|W^h[i]\right|+1)+1\right\rceil+\sum_{j=0}^{n-1}\lceil\log_2(|L[j]|+1)+1\rceil\\
&=\ 8\ \times\ n\ \times\ k\ +\ 3\ \times\ 8\ \times\ n\ +\ 3\ \times\ 8\ \times\ m\ +\ 7\ \times\ n\\
&=\ 8\ \times\ n\ \times\ k\ +\ 31\ \times\ n\ +\ 24\ \times\ m.
\end{aligned}$$

In the case of CPLEX using simplex, elliptic, and interior point algorithms, the worst-case time complexity of PBTO is as follows:

(1) Simplex: $O(n \times m\log m + n \times m) + O(2 \times n \times m + 2 \times n \times k) + O(2^{k \times n})$;
(2) Elliptic: $O(n \times m\log m + n \times m) + O(2 \times n \times m + 2 \times n \times k) + O((n \times k)^4 \times (8 \times n \times k + 31 \times n + 24 \times m))$;
(3) Interior point: $O(n \times m\log m + n \times m) + O(2 \times n \times m + 2 \times n \times k) + O((n \times k)^{3.5} \times (8 \times n \times k + 31 \times n + 24 \times m))$.

We assume that the optimal method for solving the TOEC problem is also using CPLEX. The optimal method does not reduce the size of the TOEC problem, which has the following input data: L, Q, R^0, R^1, R^2, W^0, W^1, and W^2. Therefore, the number of decision variables of the optimal method is $n \times m$, and the number of bits required for input data is $8 \times n \times m + 31 \times n + 24 \times m$. In the case of CPLEX using simplex, elliptic, and interior point algorithms, the worst-case time complexity of the optimal method is as follows:

(1) Simplex: $O(n \times m\log m + n \times m) + O(4 \times n \times m) + O(2^{n \times m})$;
(2) Elliptic: $O(n \times m\log m + n \times m) + O(4 \times n \times m) + O((n \times m)^4 \times (8 \times n \times m + 31 \times n + 24 \times m))$;
(3) Interior point: $O(n \times m\log m + n \times m) + O(4 \times n \times m) + O((n \times m)^{3.5} \times (8 \times n \times m + 31 \times n + 24 \times m))$.

In the discussed scenario, k is a constant (typically less than 10). As a result, the time complexity of the PBTO and optimal methods, as well as their time complexity ratios, can be simplified, as shown in Table 10.

Table 10. The time complexity of the PBTO, corresponding optimal methods, and their time complexity ratios.

Algorithms	PBTO	Optimal	PBTO/Optimal
simplex	$O(2^n)$	$O(2^{n \times m})$	$1/O(2^{n \times (m-1)})$
ellipsoid	$O(n^5)$	$O((n \times m)^5)$	$1/O(m^5)$
interior point	$O(n^{4.5})$	$O((n \times m)^{4.5})$	$1/O(m^{4.5})$

6. Experiments

In this section, we conducted extended experiments to simulate the task offloading process in ECC and evaluated the performance of the proposed approach and the impact of important parameters on performance. All experiments were coded in Java and performed on a Windows platform equipped with Intel Core i5-1240P @ 1.7 GHz and 16GB RAM.

6.1. Experimental Settings

To evaluate the performance of PBTO, we compare it with three benchmark methods, including an optimal method and two heuristic methods:

(1) OPT: It uses IBM CPLEX Optimizer to find the optimal solution to the optimization model introduced in Section 4 and offloads tasks based on the solution.
(2) ETO [23]: In this approach, tasks are prioritized to be offloaded at the edge as long as an ES exists with enough resources to complete the current task; otherwise, it is offloaded to the cloud.
(3) ATO [22]: In this approach, tasks are equally distributed to the edge and the cloud according to the order of submission of requests to make the edge-cloud load balanced.

ETO and ATO are two heuristic methods that employ prioritized edge and average edge-cloud policies for task offloading, respectively. As a result, ETO and ATO have low time overhead, but the quality of the obtained solutions is not high. We choose ETO and ATO as benchmark comparison methods to evaluate the effectiveness of the proposed PBTO algorithm, i.e., to compare how close they are to the optimal solution.

The efficiency and effectiveness of the PBTO algorithm are mainly affected by six key parameters: (1) the number of tasks (n); (2) the number of servers (m); (3) the proportion of

ESs (θ); (4) the proportion of large tasks (δ); (5) the number of primary-backup instances of the tasks ($L[j]$); and (6) the Top k candidate servers (k). Among them, the number of tasks and servers are two key parameters that affect the execution time of the PBTO algorithm. Moreover, the proportion of ESs affects the capacity of the available resources, and the proportion of large tasks affects the resource requirements. These two parameters, along with the number of tasks, the number of servers, the number of primary-backup instances of the tasks, and the Top k candidate servers, collectively affect the effectiveness of the PBTO algorithm, i.e., delay cost, resource cost, and total cost. Based on the six key parameters above, we conducted six sets of experiments to analyze the impact of these parameters on the efficiency and effectiveness of the PBTO algorithm. Table 11 summarizes the settings of the experiments, which will be discussed in the next section. Each experiment is repeated 100 times, and the results are then averaged.

Table 11. Experimental settings.

	n	m	Θ (%)	δ (%)	$L[j]$	k
Set #1	100, 200, . . . , 1000	50, 100, . . . , 500	50	50	[1, 3]	10
Set #2	500	100, 150, . . . , 400	50	50	[1, 3]	10
Set #3	500	250	0, 20, . . . , 100	50	[1, 3]	10
Set #4	500	250	50	0, 20, . . . , 100	[1, 3]	10
Set #5	500	250	50	50	1, 2, 3, 4, 5	10
Set #6	500	250	50	50	[1, 3]	5, 10, 15, 20, 25

From Section 4, we can find that some attributes of tasks and servers are the key parameters for evaluating the Q matrix, such as the length of the task, the size of the task's input and output data, the task's resource requirements, and the server's resource capacity and price. Therefore, in experiment Sets #1–6, we employ these parameters to create the Q matrix. In experiment Sets #1–6, tasks are categorized into two types: small tasks and large tasks. The specific parameter settings for each task type are shown in Table 12. The resource capacity and price attributes of the servers are set as shown in Table 13.

In recent years, researchers have carried out a lot of work on the task offloading problem in ECC [17,37,38], and we refer to their parameter settings in the simulation experiments and combine them with the requirements of our application scenarios to set the parameters, as shown in Tables 11–13. All the attributes outlined in Tables 12 and 13 are generated randomly within the specified range. Moreover, the objective weighting coefficient a is set as 0.5, while the signal-to-noise ratio SNR for a BS is set as 100 dB [22]. The transmission delay from the ECB to the ESs is chosen randomly from 5 to 15 ms, while the transmission delay from the ECB to the CSs varies randomly from 50 to 250 ms [17].

Table 12. Attributes of the Task Settings.

Parameter	Small Task	Large Task	Unit
Computing requirements	[200, 800]	[1000, 2000]	MIPS
Memory requirements	[0.1, 0.5]	[0.5, 1]	GB
Bandwidth requirements	[5, 20]	[20, 50]	Mbps
Task length	[500, 1000]	[1000, 4000]	MI
Input data size	[0.1, 1]	[1, 5]	MB
Output data size	[0.025, 0.25]	[0.25, 1.25]	MB

Table 13. Attributes of the Server Settings.

Parameter	Edge Server	Cloud Server	Unit
Computing capacity	[1000, 5000]	[6000, 10,000]	MIPS
Memory capacity	[2, 8]	[8, 16]	GB
Bandwidth capacity	[100, 1000]	[1000, 4000]	Mbps
CPU usage cost	[0.6, 1]	[0.2, 0.5]	G$/s
Memory usage cost	[0.04, 0.08]	[0.02, 0.06]	G$/MB
Bandwidth usage cost	[0.05, 0.1]	[0.01, 0.04]	G$/MB

6.2. Experimental Results

Figures 2–7 demonstrate the effectiveness and efficiency of all the approaches in experiment Sets #1–6, respectively, in terms of (1) the total cost (in sub-figures (a)), (2) the delay cost (in sub-figures (b)), (3) the resource cost (in sub-figures (c)), and (4) the execution time (in sub-figures (d)). In addition, to measure the relative difference in total cost and execution time between PBTO, ETO, ATO, and the optimal approach OPT, we define two metrics: total cost percent deviation and execution time percent deviation, as shown in the following equations:

$$\lambda_{\sigma}^{*} = \frac{\sigma^{*} - \sigma^{\mathrm{OPT}}}{\sigma^{\mathrm{OPT}}} \times 100 \tag{26}$$

$$\lambda_{\tau}^{*} = \frac{\tau^{\mathrm{OPT}} - \tau^{*}}{\tau^{\mathrm{OPT}}} \times 100, \tag{27}$$

where * denotes any of the approaches in PBTO, ETO, and ATO; λ_{σ}^{*} and λ_{τ}^{*} denote the total cost percent deviation and execution time percent deviation, respectively; σ^{*} and τ^{*} express the total cost and execution time of PBTO, ETO, and ATO, respectively; σ^{OPT} and τ^{OPT} indicate the total cost and execution time of OPT, respectively.

In general, OPT, being the optimal approach, clearly achieves the lowest total cost compared to all other approaches across all experiments. This comes at the cost of high time overhead and is therefore not suitable for large-scale scenarios where low latency is critical. PBTO is close to OPT in terms of total cost, delay cost, and resource cost and is much lower than OPT in terms of execution time. The other two baseline approaches have shorter execution times, but their total costs are significantly higher than those of OPT and PBTO. In conclusion, PBTO outperforms all the benchmark approaches.

(1) Impact of the number of tasks (Set #1)

In Set #1, we study the effect of different numbers of tasks on the total cost, total time delay, total monetary cost, and algorithm execution time. The ratio of the number of tasks to the number of servers, the proportions of ESs, the proportions of large tasks, the reliability requirements of tasks, and k are set to 2:1, 50%, 50%, 1–3, and 10, respectively.

As shown in Figure 2a, the total cost for all four approaches gradually increases with the number of tasks. In all cases, the total cost of OPT is minimized. PBTO removes some of the higher-cost candidate servers and then uses CPLEX to solve the RTOEC problem so that its total cost is close to that of OPT and, in most cases, almost the same as OPT. Both ETO and ATO do not focus on the optimization objective during task offloading; thus, their total cost is much higher than that of OPT and PBTO.

Figure 2b,c illustrates the effect of the number of tasks on delay cost and resource cost. In all cases, the delay cost and resource cost of PBTO and OPT are almost the same and significantly lower than those of ETO and ATO. It is worth noting that ETO has a lower delay cost than ATO due to its prioritization of offloading at the edge, but at the same time, it leads to the highest resource cost. On the contrary, ATO employs an offloading strategy that balances the edge and the cloud, and hence, it has the highest delay cost among the several approaches, but the resource cost is lower than that of ETO.

Figure 2d indicates the execution time of the four approaches. The execution time of OPT rapidly increases as the number of tasks increases, exceeding 5 s when the number of tasks is 700 and eventually reaching 12.5 s when the number of tasks is 1000. PBTO greatly

reduces the problem size by removing a large number of higher-cost candidate servers prior to offloading the tasks, and thus, its overhead time is much lower than that of OPT. Even when the number of tasks reaches 1000, PBTO takes no more than 740 ms. Both ETO and ATO only need to locally determine whether resource capacity constraints are satisfied during task offloading, so their execution time is smaller than that of OPT and PBTO.

By observing Table 14, we find that PBTO deviates minimally from OPT in total cost and substantially from OPT in execution time. In all cases, PBTO sacrifices 0.0003% of the total cost but gains an 87.84% reduction in execution time compared to OPT. ETO and ATO have much lower execution time compared to OPT. However, their total cost deviates from OPT by more than double. This indicates that our proposed PBTO is able to find a near-optimal solution to the TOEC problem at a very fast speed and is fully capable of coping with large-scale delay-sensitive task offloading scenarios.

Note that in Table 14, when n = 300, the total cost deviation value of PBTO is 0.00018, which is higher than that of n = 400 and 500. This is mainly due to the fact that when n is relatively small, the number of candidate servers for each task is less than that of cases with larger n. For instance, when n = 300 and m = 150, the number of candidate servers for each task is 150, whereas when n = 500 and m = 250, the number of candidate servers for each task is 250. Further, the PBTO algorithm removes some high-cost candidate servers during preprocessing, and this operation may erroneously remove some candidate servers that should have been present in the optimal solution. In particular, the probability of such an error is relatively high when n is small, leading to fluctuations in the value of the total cost deviation when n is small. Fortunately, although PBTO's total cost deviation is less stable for smaller n, it is still very close to the optimal solution overall.

Table 14. The percentage deviation in total cost and execution time of experiment Set #1.

λ \ n	100	200	300	400	500	600	700	800	900	1000
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0	0	0.00018	0	0	0.00014	0.00035	0.00034	0.00065	0.00136
$\lambda_{\sigma}^{\mathrm{ETO}}$	57.5	62.52	65.29	67.05	68.77	69.72	70.49	70.68	71.27	71.78
$\lambda_{\sigma}^{\mathrm{ATO}}$	72.5	81.01	84.05	85.24	86.74	87.37	88.56	88.85	89.41	90.1
$\lambda_{\tau}^{\mathrm{PBTO}}$	67.14	80.08	87.48	88.41	91.74	90.56	91.89	93.37	93.67	94.08
$\lambda_{\tau}^{\mathrm{ETO}}$	99.91	99.96	99.97	99.98	99.98	99.98	99.99	99.99	99.99	99.99
$\lambda_{\tau}^{\mathrm{ATO}}$	99.91	99.97	99.98	99.985	99.98	99.99	99.99	99.99	99.99	99.81

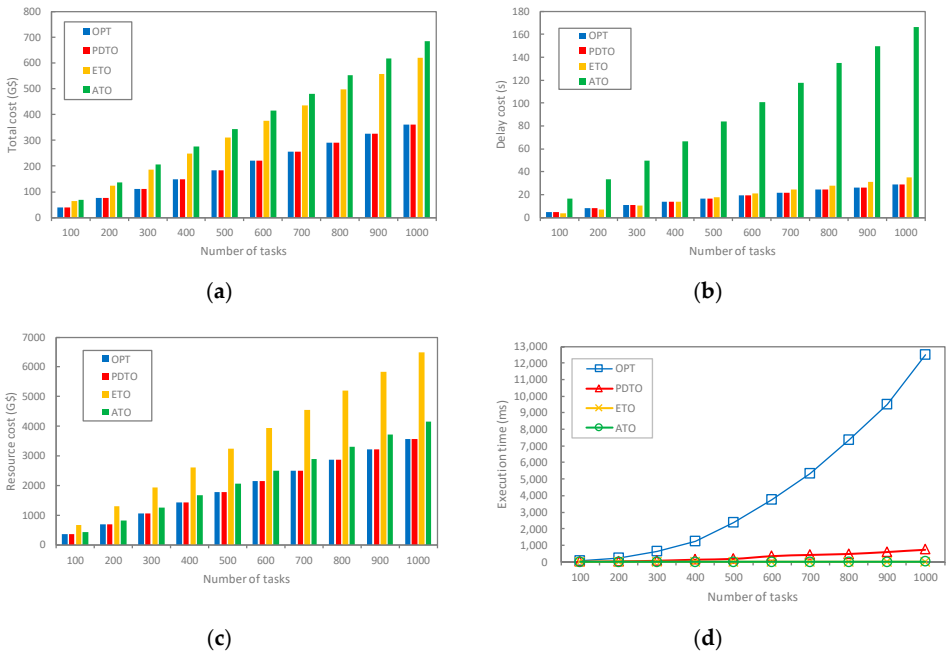


Figure 2. Experimental results of experiment Set #1 (varying number of tasks). (**a**) Total cost; (**b**) delay cost; (**c**) resource cost; (**d**) execution time.

(2) The impact of the number of servers (Set #2)

In Set #2, we investigate the impact of different numbers of servers on offloading performance with a fixed number of tasks. We fixed the number of tasks at 500 and changed the number of servers from 100 to 400, with a step of 50. The proportion of large tasks, the proportions of edge servers, the reliability requirements of tasks, and k are set to 50%, 50%, 1–3, and 10, respectively.

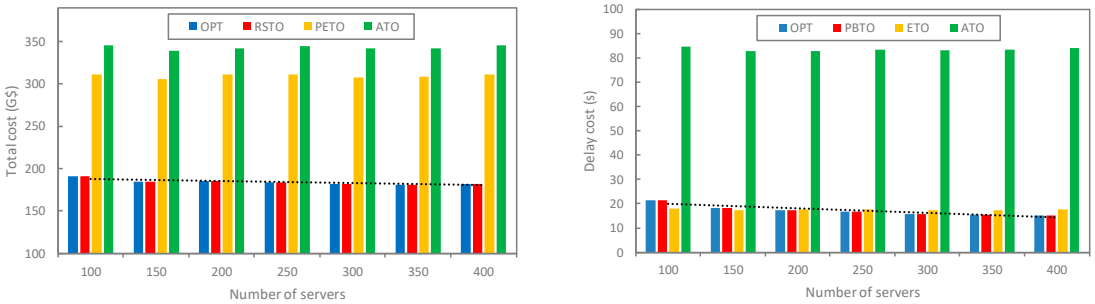
It is evident from Figure 3a that as the number of servers increases, the total cost of both OPT and PBTO shows a slight improvement, mainly because the increase in the number of servers enhances their chances of obtaining a better task-offloading solution, and it can be further confirmed from Figure 3b,c. In Figure 3b,c, as the number of servers increases, the time delay gradually decreases while the resource cost remains constant, resulting in some improvement in the total cost of both OPT and PBTO. Similar to the observations in Figure 2a–c, the total cost, delay cost, and resource cost of ETO and ATO remain stable and much higher than those of OPT and PBTO.

In Figure 3d, the execution time of OPT and PBTO increases with the number of servers, and OPT increases more rapidly than PBTO. The reason behind this phenomenon is that the increase in the number of servers leads to a rapid expansion of the search space for the TOEC problem. Since OPT and PBTO need to obtain the global optimal solution, their execution time grows accordingly. It is worth noting that PBTO's execution time does not increase as rapidly as OPT's because it reduces the search space of the problem. Similar to Figure 2d, the execution time of both ETO and ATO does not change significantly.

By observing Table 15, we find that PBTO deviates minimally from OPT in total cost and significantly from OPT in execution time. In all cases, PBTO sacrifices 0.0005% of the total cost but gains an 87.60% reduction in overall execution time compared to OPT. ETO and ATO exhibit results similar to those in Table 14, i.e., low time overhead but large total cost deviation.

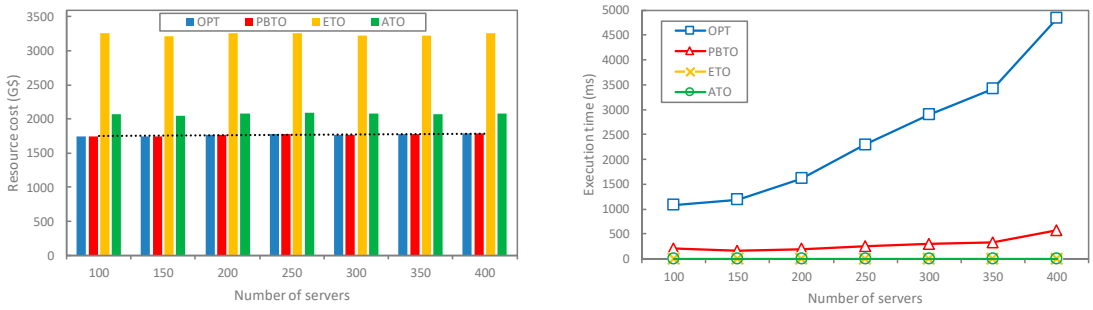
Table 15. The percentage deviation in total cost and execution time of experiment Set #2.

λ \ m	100	150	200	250	300	350	400
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0.00021	0.00157	0.0010	0.0007	0	0	0
$\lambda_{\sigma}^{\mathrm{ETO}}$	63.06	65.79	67.88	68.91	68.76	69.84	70.71
$\lambda_{\sigma}^{\mathrm{ATO}}$	80.89	83.67	84.61	86.83	88.019	88.59	89.37
$\lambda_{\tau}^{\mathrm{PBTO}}$	81.19	86.86	87.82	89.13	89.67	90.37	88.14
$\lambda_{\tau}^{\mathrm{ETO}}$	99.97	99.97	99.97	99.98	99.98	99.98	99.99
$\lambda_{\tau}^{\mathrm{ATO}}$	99.98	99.98	99.98	99.98	99.98	99.99	99.99



(**a**) (**b**)

Figure 3. *Cont.*



(**c**) (**d**)

Figure 3. Experimental results of experiment Set #2 (varying number of servers). (**a**) Total cost; (**b**) delay cost; (**c**) resource cost; (**d**) execution time. Note that the dotted lines in the graph indicate the increasing or decreasing trend of the costs.

(3) The impact of the proportions of ESs (Set #3)

In Set #3, we fix the number of tasks at 500 and increase the percentage of ESs in the total servers from 0% to 100% to compare the performance of the four approaches. The ratio of the number of tasks to the number of servers, the proportion of large tasks, the reliability requirements of tasks, and k are set to 2:1, 50%, 1–3, and 10, respectively.

From Figure 4a–c, as the percentage of ESs increases, the total cost of all approaches first improves significantly and then stabilizes. This is because when the percentage of ESs increases at the beginning, many tasks migrate from CSs to ESs, resulting in a rapid decrease in time delay and a gradual increase in resource cost, which reduces the total cost. However, as the proportion of ESs increases further, no more tasks will migrate to the ESs, and therefore the total cost stabilizes. Notably, ETO prioritizes offloading tasks at ESs, and once the ESs take some percentage, it migrates as many tasks as possible to ESs, and hence, its total cost, delay cost, and resource cost change process is more pronounced than the other three approaches. Furthermore, ACO does not have a feasible solution when all servers are ESs because of its edge-cloud balanced offloading strategy.

Figure 4d demonstrates the variation in execution time for all approaches. OPT still takes a long time to obtain the optimal solution, and the execution time of PBTO is slightly higher than that of ETO and ACO but still much lower than that of OPT. Moreover, when the percentage of ESs is close to 100%, many ESs have difficulty offloading more tasks due to resource capacity constraints, which results in OPT and PBTO taking a longer time to find a solution.

Observing Table 16, we find that PBTO deviates minimally from OPT in total cost and considerably from OPT in execution time. In all cases, PBTO sacrifices 0.0004% of the total cost but gains an 87.75% reduction in overall execution time compared to OPT. ETO and ATO yield results similar to those in Table 14, i.e., low time overhead but large total cost deviation.

Table 16. The percentage deviation in total cost and execution time of experiment Set #3.

λ \ θ(%)	0	20	40	60	80	100
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0.00112	0.00089	0.00059	0	0	0
$\lambda_{\sigma}^{\mathrm{ETO}}$	79.55	63.80	67.76	87.64	71.50	71.93
$\lambda_{\sigma}^{\mathrm{ATO}}$	79.55	81.77	85.62	88.19	89.64	
$\lambda_{\tau}^{\mathrm{PBTO}}$	90.43	90.07	88.73	87.36	86.69	83.25
$\lambda_{\tau}^{\mathrm{ETO}}$	99.98	99.98	99.98	99.98	99.98	99.98
$\lambda_{\tau}^{\mathrm{ATO}}$	99.98	99.98	99.99	99.98	99.98	

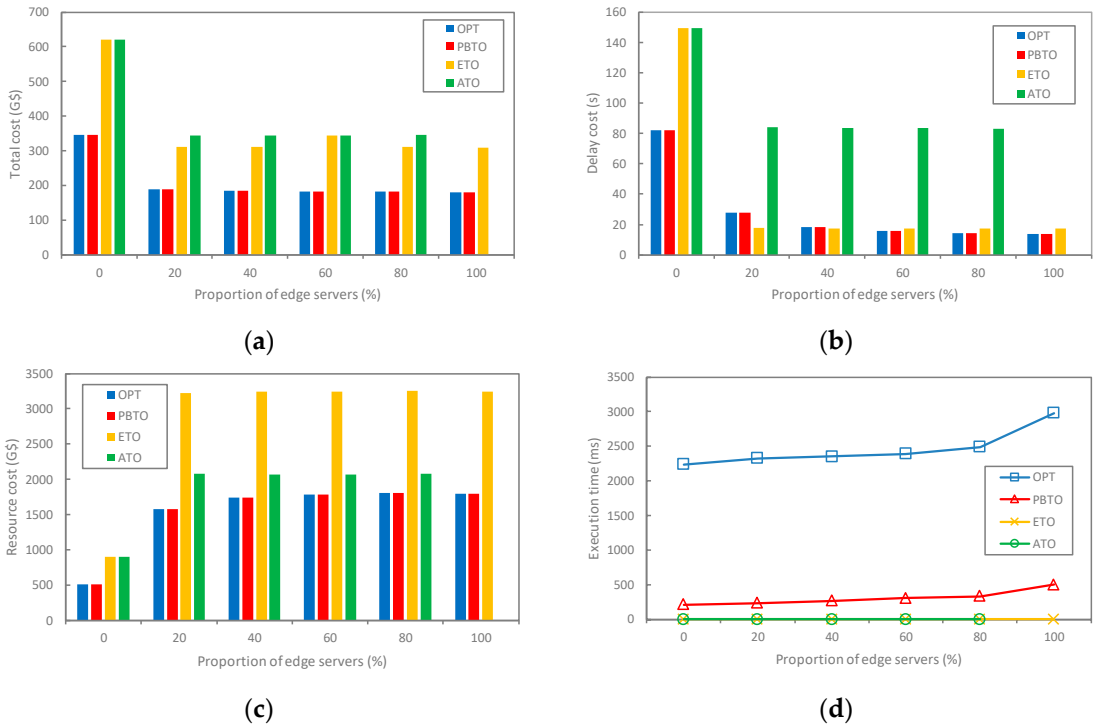


Figure 4. Experimental results of experiment Set #3 (varying proportions of ESs). (**a**) Total cost; (**b**) delay cost; (**c**) resource cost; (**d**) execution time.

(4) The impact of the proportions of large tasks (Set #4)

In set #4, we research the effect of the proportions of different task types, from all small tasks to all large tasks, on offloading performance. The total number of tasks and servers is fixed at 500 and 250, respectively. The proportion of ESs, the reliability requirements of the tasks, and k are set to be 50%, 1–3, and 10, respectively.

Figure 5a demonstrates that the total cost of all approaches increases with the proportion of large tasks. This is because large tasks result in much greater time delay and resource cost than small tasks, regardless of whether they are offloaded to ESs or CSs. From Figure 5b,c, it can be noticed that the delay cost and resource cost of all approaches increase with the proportion of large tasks. Moreover, in all cases, the total cost, delay cost, and resource cost of PBTO still closely follow OPT and outperform the other two approaches. In Figure 5d, there is no significant difference in the change in execution time of all approaches with the proportion of large tasks increasing.

Observing Table 17, we find that PBTO deviates minimally from OPT in total cost and significantly from OPT in execution time. In all cases, PBTO sacrifices 0.0008% of total cost but gains an 89.90% reduction in overall execution time compared to OPT. ETO and ATO show similar results to those in Table 14, i.e., low time overhead but large total cost deviation.

Table 17. The percentage deviation in total cost and execution time of experiment Set #4.

λ \ δ(%)	0	20	40	60	80	100
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0	0	0	0.00051	0.00282	0.00219
$\lambda_{\sigma}^{\mathrm{ETO}}$	76.67	74.53	70.59	67.30	64.67	62.35
$\lambda_{\sigma}^{\mathrm{ATO}}$	98.75	95.35	91.04	82.28	76.07	70.64
$\lambda_{\tau}^{\mathrm{PBTO}}$	89.22	89.54	89.50	90.19	90.49	90.49
$\lambda_{\tau}^{\mathrm{ETO}}$	99.98	99.98	99.97	99.96	99.98	99.98
$\lambda_{\tau}^{\mathrm{ATO}}$	99.98	99.98	99.98	99.98	99.98	99.98

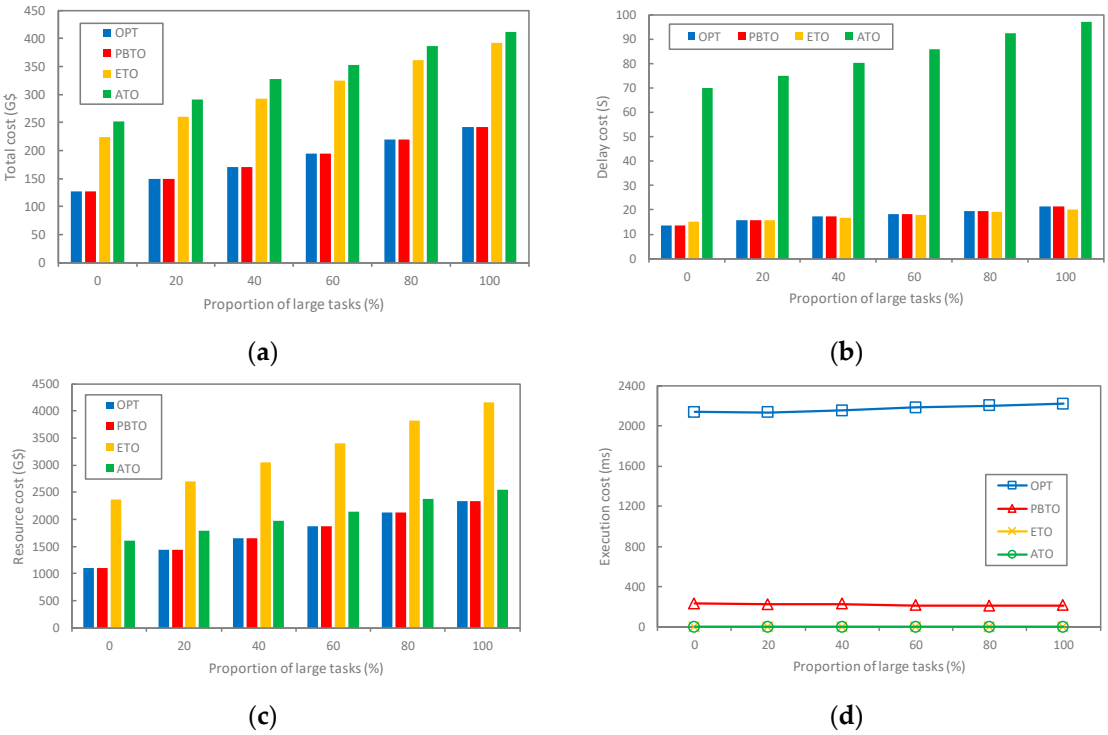


Figure 5. Experimental results of experiment Set #4 (varying proportions of large tasks). (**a**) Total cost; (**b**) delay cost; (**c**) resource cost; (**d**) execution time.

(5) The impact of the reliability requirements of the tasks (Set #5)

In set #5, we compare the performance of four approaches while increasing the number of primary-backup task instances from 1 to 5. The total number of tasks and servers is fixed at 500 and 250, respectively. The proportion of ESs, the proportion of task types, and k are set to 50%, 50%, and 10, respectively.

Figure 6a shows that the total cost of all approaches increases with the number of task primary backups. For a task, each additional backup, whether offloading it to ES or CS, incurs a certain time delay and resource cost. As shown in Figure 6b,c, the delay cost and resource cost of all four approaches increase with the number of task primary backups.

Figure 6d illustrates that as $L[j]$ increases from 1 to 5, the execution time of OPT first decreases rapidly and then gradually increases. The possible reason is that when $L[j] = 1$, there are more assignable servers for each task, and the OPT takes more time to obtain the optimal solution. When $L[j]$ increases to 2, it is equivalent to reducing the number of assignable servers for each task by half. Thus, the execution time of the OPT decreases rapidly. However, when $L[j]$ continues to increase, although the number of assignable servers decreases, the number of primary-backup instances that need to be offloaded increases, so the execution time of OPT starts to increase gradually again. The execution time of PBTO changes similarly to that of OPT, but the execution time is much smaller than that of OPT, similar to the reason explained in Figure 2d. The time changes of ETO and ACO are very slow, and there is no significant difference.

Observing Table 18, we find that PBTO deviates minimally from OPT in total cost and substantially from OPT in execution time. In all cases, PBTO sacrifices 0.0006% of the total cost but gains an 82.91% reduction in overall execution time compared to OPT. ETO and ATO yield results similar to those in Table 14, i.e., low time overhead but large total cost deviation.

Table 18. The percentage deviation in total cost and execution time of experiment Set #5.

λ \ $L[j]$	1	2	3	4	5
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0.00077	0.00049	0.00212	0	0
$\lambda_{\sigma}^{\mathrm{ETO}}$	70.89	69.18	67.67	66.67	65.36
$\lambda_{\sigma}^{\mathrm{ATO}}$	87.43	87.73	85.70	84.39	83.05
$\lambda_{\tau}^{\mathrm{PBTO}}$	83.77	82.59	82.11	83.26	82.83
$\lambda_{\tau}^{\mathrm{ETO}}$	99.98	99.97	99.96	99.97	99.97
$\lambda_{\tau}^{\mathrm{ATO}}$	99.98	99.95	99.97	99.98	99.98

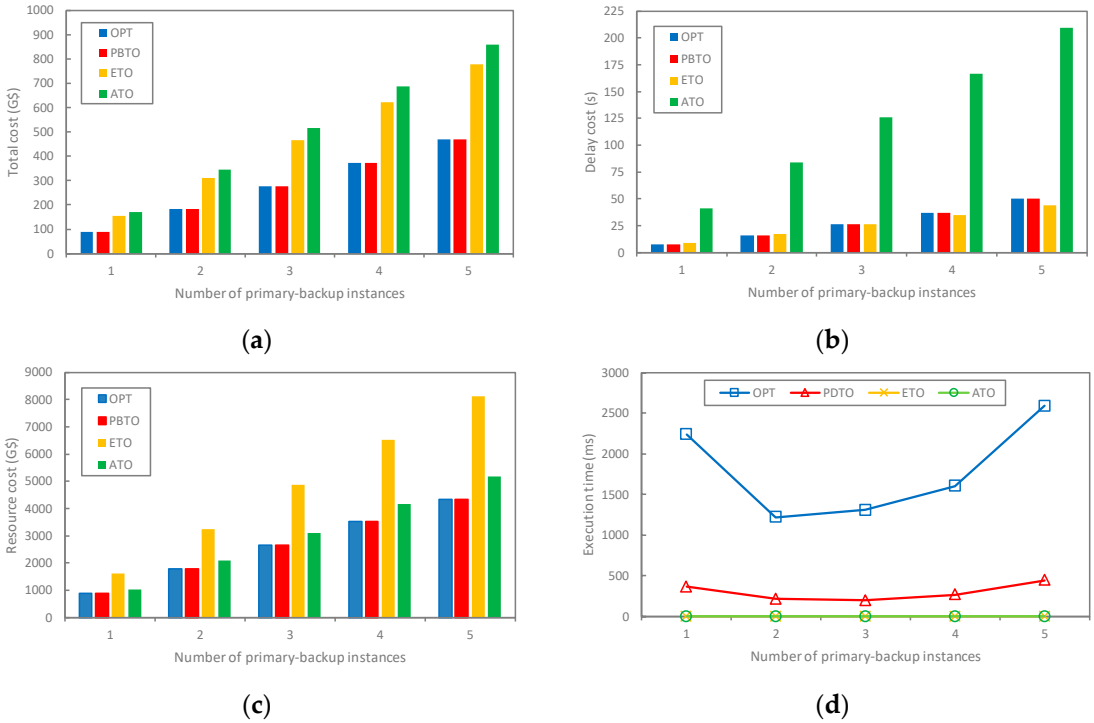


Figure 6. Experimental results of experiment Set #5 (varying reliability requirements of the tasks). (**a**) Total cost; (**b**) delay cost; (**c**) resource cost; (**d**) execution time.

(6) The impact of the k (Set #6)

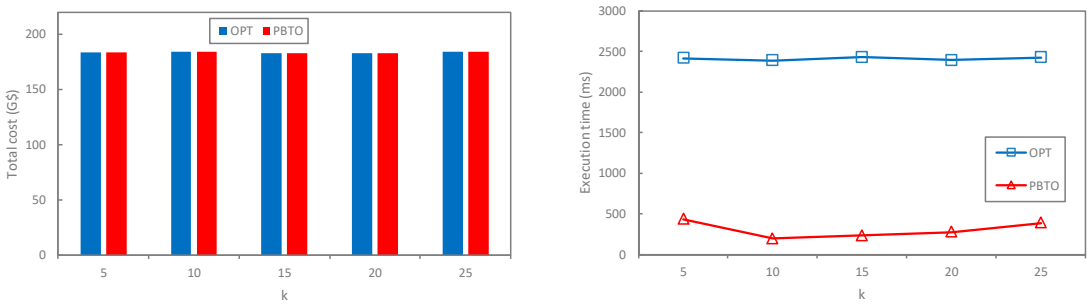
In set #6, we evaluated the effects of k on the total cost and execution time of PBTO. Parameters n and m are fixed at 500 and 250, respectively. The reliability requirements of the tasks, the proportion of large tasks, and the proportion of ESs were set to 1–3, 50%, and 50%, respectively.

Figure 7a shows that the total cost of PBTO does not change much as k increases. However, in Figure 7b, the execution time of PBTO decreases slightly and then increases gradually with the increase in k. The reason may be that when the number of candidate servers is relatively small, PBTO takes more time to obtain a feasible solution, and the execution time decreases as the number of candidate servers gradually increases. Nevertheless, when the number of candidate servers increases further, PBTO takes more time to obtain the optimal solution from the many feasible solutions. It is worth noting that although the execution time of PBTO increases with the increase of k, the overall growth trend is relatively flat, and even at k = 25, the execution time is only 385 ms, which is acceptable for general IoT tasks. Additionally, increasing the value of k does not improve the total cost much, as shown in Figure 7a; the total cost remains around 183 G$ for all cases. Therefore, it is not necessary to set the value of k too large. In our scenario, setting k to 10 is appropriate. Since OPT does not reduce the number of candidate servers, its execution time is not affected by k.

Observing Table 19, we find that PBTO deviates minimally from OPT in total cost and significantly from OPT in execution time. In all cases, PBTO sacrifices 0.0004% of the total cost but gains an 87.36% reduction in overall execution time compared to OPT.

Table 19. The percentage deviation in total cost and execution time of experiment Set #6.

λ \ n	5	10	15	20	25
$\lambda_{\sigma}^{\mathrm{PBTO}}$	0	0	0.00010	0.00038	0.00065
$\lambda_{\tau}^{\mathrm{PBTO}}$	82.07	91.74	90.48	88.41	84.09



(**a**) (**b**)

Figure 7. Experimental results of experiment Set #6 (varying k). (**a**) Total cost; (**b**) execution time.

7. Conclusions

In an ECC system, offloading tasks to ESs or ESs quickly and efficiently is an important yet challenging problem. In this paper, we propose a task offloading method for heterogeneous scenarios involving multi-tasks, multi-ESs, and multi-CSs, which takes minimizing delay cost and resource cost as the optimization objective while considering the reliability requirements of tasks and resource capacity constraints of servers. First, we formally define the TOEC problem by extending GMRA and formulating it as an ILP. Then, we design the PBTO algorithm to obtain a near-optimal solution quickly. Further, we carry out a series of simulation experiments comparing the PBTO algorithm with the optimal method and two heuristic methods.

The experimental results show that the execution time of the proposed PBTO algorithm is reduced by 87.23%, while the total cost increases by only 0.0004% compared to the optimal method. The two heuristic methods, although better than PBTO in terms of time performance, have much lower-quality solutions. In all cases, the total cost of both heuristic methods is more than double that of the optimal method, making them difficult to apply in real-world IoT task offloading scenarios.

In conclusion, the proposed method is suitable for large-scale IoT task offloading scenarios with strict low-delay requirements and can also be applied to other related scenarios, such as crowdsourcing task allocation, micro-services deployment, drone swarm scheduling, and task offloading in vehicle edge computing.

However, our approach does not take into account the dynamically changing requirements of resource prices and network conditions and ignores the offloading scenario where an IoT task is composed of multiple interdependent subtasks. Our future research will focus on the following aspects:

(1) Extending the task offloading model with the ability to characterize energy consumption and explore the impact of dynamically changing network conditions on delay, resource cost, and energy consumption.
(2) Considering a scenario where service providers dynamically change resource prices over time and demand in pursuit of greater benefits, we will study the time-varying multidimensional resource dynamic pricing model and design corresponding task offloading algorithms.

(3) In real-world application scenarios, an IoT task generally consists of multiple subtasks, and we will study how to divide the task into multiple subtasks reasonably and propose an offloading algorithm in the edge-cloud collaboration environment based on the dependencies between subtasks.
(4) More role assignment models, such as GRA+ [38,45] and GRA++ [39,45], can be applied to IoT and edge-cloud problem-solving.

Author Contributions: Conceptualization, L.L. and H.Z.; methodology, L.L. and H.Z.; software, L.L. and T.W.; validation, L.L. and M.T.; formal analysis, L.L. and H.Z.; investigation, M.T. and T.W.; resources, H.Z.; data curation, M.T.; writing—original draft preparation, L.L.; writing—review and editing, L.L. and H.Z.; visualization, T.W.; supervision, H.Z.; project administration, L.L.; funding acquisition, M.T. and H.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (Grant No. RGPIN-2018-04818), the Social Sciences and Humanities Research Council of Canada (SSHRC) Insight Grant (Grant No. 435-2023-1056), the National Social Science Fund of China (Grant No. 23AJY018), the Significant Project of Jiangsu College Philosophy and Social Sciences Research (Grant No. 2021SJZDA153), and the Planning Fund Project of Humanities and Social Sciences Research of Ministry of Education (Grant No. 23YJA870009).

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank the editor and the anonymous reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Siow, E.; Tiropanis, T.; Hall, W. Analytics for the Internet of Things: A Survey. *ACM Comput. Surv.* **2018**, *51*, 1–36. [CrossRef]
2. Al-Shuwaili, A.N.; Simeone, O. Energy-efficient Resource Allocation for Mobile Edge Computing-based Augmented Reality Applications. *IEEE Wirel. Commun. Lett.* **2017**, *6*, 398–401. [CrossRef]
3. Fortino, G.; Savaglio, C.; Spezzano, G.; Zhou, M. Internet of Tings as System of Systems: A Review of Methodologies, frameworks, Platforms, and Tools. *IEEE Trans. Syst. Man Cybern. Syst.* **2021**, *51*, 223–236. [CrossRef]
4. Cicconetti, C.; Conti, M.; Passarella, A. A Decentralized Framework for Serverless Edge Computing in the Internet of Things. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 2166–2180. [CrossRef]
5. Xu, J.; Chen, L.; Zhou, P. Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM'2018), Honolulu, HI, USA, 16–19 April 2018; IEEE: New York, NY, USA, 2018; pp. 207–215.
6. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surveys Tuts.* **2017**, *19*, 2322–2358. [CrossRef]
7. Lin, L.; Liao, X.; Jin, H.; Li, P. Computation Offloading Toward Edge Computing. *Proc. IEEE.* **2019**, *107*, 1584–1607. [CrossRef]
8. Fernando, N.; Loke, S.W.; Rahayu, W. Computing with Nearby Mobile Devices: A Work Sharing Algorithm for Mobile Edge-Clouds. *IEEE Trans. Cloud Comput.* **2019**, *7*, 329–343. [CrossRef]
9. Han, Z.; Tan, H.; Li, X.-Y.; Jiang, S.H.C.; Li, Y.; Lau, F.C. OnDisc: Online Latency-Sensitive Job Dispatching and Scheduling in Heterogeneous Edge-Clouds. *IEEE/ACM Trans. Netw.* **2019**, *27*, 2472–2485. [CrossRef]
10. Peng, C.; Zhu, H.; Liu, L.; Grewal, R. Optimal Data Allocation in the Environment of Edge and Cloud Servers. In Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC'2022), Shanghai, China, 15–18 December 2022; pp. 1–6.
11. Li, R.; Zhou, Z.; Chen, X.; Ling, Q. Resource Price-Aware Offloading for Edge-Cloud Collaboration: A Two-Timescale Online Control Approach. *IEEE Trans. Cloud Comput.* **2022**, *10*, 648–661. [CrossRef]
12. Zhang, P.; Zhou, M.; Fortino, G. Security and Trust Issues in Fog Computing: A Survey. *Future Gener. Comput. Syst.* **2018**, *88*, 16–27. [CrossRef]
13. Liu, L.; Zhu, H.; Chen, S.; Huang, Z. Privacy Regulation Aware Service Selection for Multi-Provision Cloud Service Composition. *Future Gener. Comput. Syst.* **2022**, *126*, 263–278. [CrossRef]
14. Yang, S.; Li, F.; Trajanovski, S.; Yahyapour, R.; Fu, X. Recent Advances of Resource Allocation in Network Function Virtualization. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 295–314. [CrossRef]
15. Xie, G.; Zeng, G.; Chen, Y.; Bai, Y.; Zhou, Z.; Li, R.; Li, K. Minimizing Redundancy to Satisfy Reliability Requirement for a Parallel Application on Heterogeneous Service-Oriented Systems. *IEEE Trans. Serv. Comput.* **2020**, *13*, 871–886. [CrossRef]
16. Zhou, B.; Dastjerdi, A.V.; Calheiros, R.N.; Buyya, R. An Online Algorithm for Task Offloading in Heterogeneous Mobile Clouds. *ACM Trans. Internet Technol.* **2018**, *18*, 1–25. [CrossRef]

17. Hoseiny, F.; Azizi, S.; Shojafar, M.; Tafazolli, R. Joint QoS-Aware and Cost-Efficient Task Scheduling for Fog-Cloud Resources in a Volunteer Computing System. *ACM Trans. Internet Technol.* **2021**, *21*, 1–21. [CrossRef]
18. Dai, F.; Liu, G.; Mo, Q.; Xu, W.; Huang, B. Task Offloading for Vehicular Edge Computing with Edge-Cloud Cooperation. *World Wide Web.* **2022**, *25*, 1999–2017. [CrossRef]
19. Li, J.; Liang, W.; Xu, W.; Xu, Z.; Jia, X.; Zhou, W.; Zhao, J. Maximizing User Service Satisfaction for Delay-Sensitive IoT Applications in Edge Computing. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 1199–1212. [CrossRef]
20. Tang, T.; Li, C.; Liu, F. Collaborative Cloud-Edge-End Task Offloading with Task Dependency Based on Deep Reinforcement Learning. *Comput. Commun.* **2023**, *209*, 78–90. [CrossRef]
21. Mazouzi, H.; Boussetta, K.; Achir, N. Maximizing Mobiles Energy Saving Through Tasks Optimal Offloading Placement in Two-Tier Cloud: A Theoretical and An Experimental Study. *Comput. Commun.* **2019**, *144*, 132–148. [CrossRef]
22. Su, Q.; Zhang, Q.; Li, W.; Zhang, X. Primal-Dual-Based Computation Offloading Method for Energy-Aware Cloud-Edge Collaboration. *IEEE Trans. Mobile Comput.* 2023, *to be published*. [CrossRef]
23. Liu, T.; Fang, L.; Zhu, Y.; Tong, W.; Yang, Y. A Near-Optimal Approach for Online Task Offloading and Resource Allocation in Edge-Cloud Orchestrated Computing. *IEEE Trans. Mob. Comput.* **2022**, *21*, 2687–2700. [CrossRef]
24. Xu, X.; Liu, Q.; Luo, Y.; Peng, K.; Zhang, X.; Meng, S.; Qi, L. A Computation Offloading Method over Big Data for IoT Enabled Cloud-Edge Computing. *Future Gener. Comput. Syst.* **2019**, *95*, 522–533. [CrossRef]
25. Laili, Y.; Guo, F.; Ren, L.; Li, X.; Li, Y.; Zhang, L. Parallel Scheduling of Large-Scale Tasks for Industrial Cloud-Edge Collaboration. *IEEE Internet Things J.* **2023**, *10*, 3231–3242. [CrossRef]
26. Long, S.; Zhang, Y.; Deng, Q.; Pei, T.; Ouyang, J.; Xia, Z. An Efficient Task Offloading Approach Based on Multi-Objective Evolutionary Algorithm in Cloud-Edge Collaborative Environment. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 645–657. [CrossRef]
27. Haber, E.E.; Nguyen, T.M.; Assi, C. Joint Optimization of Computational Cost and Devices Energy for Task Offloading in Multi-Tier Edge-Clouds. *IEEE Trans. Commun.* **2019**, *67*, 3407–3421. [CrossRef]
28. Wang, J.; Liu, K.; Li, B.; Liu, T.; Li, R.; Han, Z. Delay-Sensitive Multi-Period Computation Offloading with Reliability Guarantees in Fog Networks. *IEEE Trans. Mob. Comput.* **2020**, *19*, 2062–2075. [CrossRef]
29. Ghanavati, S.; Abawajy, J.H.; Izadi, D. Automata-Based Dynamic Fault Tolerant Task Scheduling Approach in Fog Computing. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 488–499. [CrossRef]
30. Hou, X.; Ren, Z.; Wang, J.; Zheng, S.; Cheng, W.; Zhang, H. Distributed Fog Computing for Latency and Reliability Guaranteed Swarm of Drones. *IEEE Access* **2020**, *8*, 7117–7130. [CrossRef]
31. Dong, L.; Wu, W.; Guo, Q.; Satpute, M.N.; Znati, T.; Du, D. Reliability-Aware Offloading and Allocation in Multilevel Edge Computing System. *IEEE Trans. Reliab.* **2021**, *70*, 200–211. [CrossRef]
32. Liang, J.; Ma, B.; Feng, Z.; Huang, J. Reliability-Aware Task Processing and Offloading for Data-Intensive Applications in Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2023**, *4*, 4668–4680. [CrossRef]
33. Siyadatzadeh, R.; Mehrafrooz, F.; Ansari, M.; Safaei, B.; Shafique, M.; Henkel, J.; Ejlali, A. ReLIEF: A Reinforcement-Learning-Based Real-Time Task Assignment Strategy in Emerging Fault-Tolerant Fog Computing. *IEEE Internet Things, J.* **2023**, *10*, 10752–10763. [CrossRef]
34. Yao, J.; Ansari, N. Fog Resource Provisioning in Reliability-Aware IoT Networks. *IEEE Internet Things J.* **2019**, *6*, 8262–8269. [CrossRef]
35. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [CrossRef]
36. Li, J.; Liang, W.; Xu, W.; Xu, Z.; Jia, X.; Zomaya, A.Y.; Guo, S. Budget-Aware User Satisfaction Maximization on Service Provisioning in Mobile Edge Computing. *IEEE Trans. Mobile Comput.* **2022**, *22*, 7057–7069. [CrossRef]
37. Dabiri, S.; Azizi, S.; Abdollahpouri, A. Optimizing Deadline Violation Time and Energy Consumption of IoT Jobs in Fog-Cloud Computing. *Neural Comput. Appl.* **2022**, *34*, 21157–21173. [CrossRef]
38. Azizi, S.; Othman, M.; Khamfroush, H. DECO: A Deadline-Aware and Energy-Efficient Algorithm for Task Offloading in Mobile Edge Computing. *IEEE Syst. J.* **2023**, *17*, 952–963. [CrossRef]
39. Zhang, Z.; Wang, N.; Wu, H.; Tang, C.; Li, R. MR-DRO: A Fast and Efficient Task Offloading Algorithm in Heterogeneous Edge/Cloud Computing Environments. *IEEE Internet Things J.* **2023**, *10*, 3165–3178. [CrossRef]
40. Zhu, H.; Zhou, M. Role-Based Collaboration and Its Kernel Mechanisms. *IEEE Trans. Syst. Man Cybern. Syst.* **2006**, *36*, 578–589.
41. Zhu, H. Group Role Assignment with Constraints (GRA+): A New Category of Assignment Problems. *IEEE Trans. Syst. Man Cybern. Syst.* **2023**, *53*, 1670–1683. [CrossRef]
42. Zhu, H. Maximizing Group Performance While Minimizing Budget. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *50*, 633–645. [CrossRef]
43. Zhu, H.; Liu, D.; Zhang, S.; Teng, S.; Zhu, Y. Solving the Group Multirole Assignment Problem by Improving the ILOG Approach. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *47*, 3418–3424. [CrossRef]
44. Burkard, R.; Amico, M.D.; Martello, S. *Assignment Problems: Revised Reprint*; SIAM: Philadelphia, PA, USA, 2012.
45. IBM. IBM ILOG CPLEX Optimizer. 2023. Available online: https://www.ibm.com/analytics/cplex-optimizer (accessed on 1 September 2023).
46. Zhu, H. *E-CARGO and Role-Based Collaboration: Modeling and Solving Problems in the Complex World*; Wiley-IEEE Press: Hoboken, NJ, USA, 2021.

47. Schrijver, A. Theory of Linear and Integer Programming. In *Wiley-Interscience Series in Discrete Mathematics and Optimization*; Wiley Press: Hoboken, NJ, USA, 1999; Volume I–XI, pp. 1–47.
48. Kallberg, J.G. A Generalized Version of Shannon's Theorem. *Br. J. Dermatol.* **1974**, *151*, 565–570.
49. Peng, Q.; Wu, C.; Xia, Y.; Ma, Y.; Wang, X.; Jiang, N. Dosra: A Decentralized Approach to Online Edge Task Scheduling and Resource Allocation. *IEEE Internet Things J.* **2022**, *9*, 4677–4692. [CrossRef]
50. Potraa, F.A.; Wright, S.J. Interior-Point Methods. *J. Comput. Appl. Math.* **2000**, *124*, 281–302. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.