symmetry

MDPI

Article

Water Pumping and Refilling (WPR): A Resource Allocation Algorithm for Maximizing Acceptance Ratio in Asymmetrical Edge Computing Networks

Li Dong *, Wenji He and Yunjie Liu

School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China
* Correspondence: donglisci@gmail.com

Abstract: Computation offloading has received a significant amount of attention in recent years, with many researchers proposing joint offloading decision and resource allocation schemes. However, although existing delay minimization schemes achieve minimum delay costs, they do so at the cost of losing possible further maximization of the number of serviced requests. Furthermore, the asymmetry between uplink and downlink poses challenges to resource allocation in edge computing. This paper addresses this issue by formulating the joint computation offloading and edge resource allocation problem as a mixed-integer nonlinear programming (MINLP) problem in an edge-enabled asymmetrical network. Leveraging the margin between a delay-minimum scheme and a near-deadline scheme, a water pumping and refilling (WPR) algorithm is proposed to maximize the number of accepted requests. The WPR algorithm can function both as a supplementary algorithm to a given offloading scheme and as a standalone algorithm to obtain a resource allocation scheme following a customizable refilling policy. The simulation results demonstrated that the proposed algorithm outperforms delay-minimum schemes in achieving a high acceptance ratio.

Keywords: computation offloading; mobile edge computing; resource allocation; delay minimum; WPR

Citation: Dong, L.; He, W.; Liu, Y. Water Pumping and Refilling (WPR): A Resource Allocation Algorithm for Maximizing Acceptance Ratio in Edge Computing System. *Symmetry* **2023**, *15*, 985. https://doi.org/10.3390/sym15050985

Academic Editor: Silvio Pardi

Received: 2 March 2023
Revised: 27 March 2023
Accepted: 19 April 2023
Published: 26 April 2023

1. Introduction

Computation offloading has emerged as a prominent research area in edge computing-enabled systems in recent years [1]. Its fundamental concept involves leveraging the computing resources deployed at the network edge to offer faster and more satisfactory computational services to user devices (UDs) [2,3]. However, the burgeoning number of Internet of Things [4] devices exacerbates the already constrained computing resources of edge nodes. When an asymmetry edge node is overloaded, it is unable to accommodate all the computation offloading requests. Therefore, how to allocate resources to maximize the number of serviced requests in asymmetrical systems, while respecting the given resource limitations, remains a crucial issue.

As computation offloading requests arrive, the BS decides which requests should be accepted and how many resources should be allocated for the accepted requests. Different objectives lead to different resource allocation schemes. The primary objective of current research in computation offloading is to minimize delay or energy consumption [5]. In situations where multiple users demand offloading services, it is common to minimize the weighted sum or the sum of users' processing delays [6,7]. However, weight setting is typically based on empirical methods, and schemes with different weightings may yield distinct performances. Although designs that minimize delay cost lead to attractive resource allocation schemes, most of their resource allocation results tend to be over-provisioned to some extent. On the other hand, energy consumption minimization schemes, from the perspective of user devices, extend the battery life of UDs and achieve energy efficiency

at the system level [8]. However, such schemes may not be optimal from the perspective of network operators (NOs) because they fail to accommodate additional computation offloading requests, thereby reducing the economic revenues of NOs.

Several existing studies propose incentive-driven computation offloading and resource allocation schemes to benefit edge service providers [9]. For example, Yuan et al. [10] aimed to maximize the profits of remote clouds by accepting more requests without causing high bandwidth consumption and energy consumption for task processing. However, such schemes may not be suitable for edge servers with limited resources, particularly when facing excessive terminal requests. Under such circumstances, the edge server may be forced to exhaust its available resources to accommodate end requests, which involves both request acceptance and resource allocation. Therefore, a joint request acceptance and resource allocation scheme with maximized accepted requests is necessary.

Although these designs function well when there are sufficient resources in the serving network, the focus of request acceptance and resource allocation should shift towards utilizing constrained resources to accept as many requests as possible when the resource is insufficient. Considering this, from the user's perspective, a satisfactory service does not necessarily need have the shortest service delay, as long as a certain level of service agreement [11] is met. Therefore, NOs do not have to distribute the network resource for a system-wide delay minimal objective, which releases the network's potential to accommodate more requests. In this regard, this paper proposes a joint computation offloading acceptance and edge resource allocation scheme to maximize the number of accepted requests in an edge-enabled asymmetrical network. Additionally, a scalable water pumping and refilling algorithm is proposed to accommodate requests on the basis of the aforementioned delay-minimal schemes. The primary innovations of this paper are summarized as follows:

- A low-complexity water pumping and refilling (WPR) algorithm is proposed to release the untapped potential of the network and accommodate more requests, based on the delay cost minimization scheme. This approach can serve as both a supplementary method and a standalone method when combined with a specific customization strategy.
- The joint computation offloading and edge resource allocation problem is formulated as a mixed integer nonlinear programming problem with the objective of maximizing the number of accepted requests. Resource margins between the delay cost minimization scheme and the desirable quality of service (QoS) scheme are exploited to accommodate more requests.
- We evaluate the performance of the proposed algorithm under various conditions. The simulation results demonstrate that our WPR algorithm outperforms the delay-cost-minimum-based schemes regarding acceptance ratio.

The remainder of this paper is organized as follows. In Section 2, we review related work on computation offloading and resource allocation in edge computing systems. In Section 3, we present the system model and relevant assumptions. The proposed water pumping and refilling algorithm is introduced in Section 4. In Section 5, we analyze and discuss the simulation results. Finally, our paper is concluded in Section 6.

2. Related Work

Among the delay-minimal schemes, Ren et al. [7] proposed a partial offloading model that divides a computational task into two parts and the divided tasks are collaboratively processed by an edge server and a remote cloud. A closed-form task splitting ratio and resource allocation scheme were provided. Similarly, Wang et al. [12] aimed to minimize task duration while meeting energy consumption constraints by allocating bandwidth equally to connected offloading user devices, using the alternating direction method of multipliers (ADMM) algorithm to determine computation node selection and computing resource allocation schemes. Wei et al. [13] selected the joint computation node, decided on the content caching, and determined the resource allocation (including radio bandwidth

and computing resources) with a two-hidden-layer deep neural network to minimize end-to-end delay for computation offloading and content delivery services, where the channel state and resource allocation are discretized. Although these schemes have achieved stunning performance, they still fall into the category of delay-minimal schemes, which means resource allocation results could be fine-tuned to accept more requests within a deadline.

Similarly, many research efforts have been made to devise energy cost minimization (ECM) schemes [14,15]. For example, Wang et al. [5] formulated EM and delay cost minimization (DCM) problems for single-user partial computation offloading, using the dynamic voltage scaling technique [16]. They optimized transmission power, computation resource allocation, and offloading ratio and reached the insightful conclusion that if a task has a stringent delay requirement (less than a threshold), it cannot be processed in a partitioned way. In [17], requests from representative locations are grouped, and computation results for requests with duplicated inputs selectively cached. They formulated the cache decision, bandwidth allocation, and computing resource allocation problem as an MINLP problem, aiming to minimize the energy consumption of the base station (BS) and all users. In [18], the deep deterministic policy gradient (DDPG) algorithm is adopted to solve the joint computation node selection and computing resource allocation problem, aiming to minimize system energy consumption. They constructed an SBS–MBS three-layer offloading model for delay-stringent tasks. Apart from the cloud- and edge-enabled processing models, some researchers supplement user devices to enhance performance. For instance, Huang et al. [19] proposed an edge-end cooperation scheme where mobile devices act as computing servers to minimize the energy consumed by the mobile devices. In [20], a three-node computation offloading scenario was studied in which the user device near the access point (AP) is exploited to relay and compute the task of the far user device with the aim of minimizing the energy consumed by the AP in a wireless powered system.

Several studies have investigated resource allocation in computation offloading from different perspectives. In [21] , the authors assumed adequate bandwidth between vehicles and the associated MEC server and adopted a Q-value-based deep reinforcement learning method to maximize the acceptance rate in vehicular networks. Zhou and Hu [22] aimed to maximize the ratio of processed bits to the energy consumed by energy-harvesting user devices for non-orthogonal multiple access and time division multiple access systems. Mukherjee et al. [23] focused on maximizing system revenue by analyzing the pricing strategy for offloaded tasks with different time constraints. Yan et al. [24] formulated the DCM and revenue maximization problem as a two-stage game, where computing resources at BS were equally allocated. Wang et al. [25] designed an online auction mechanism to maximize the profit of resource providers in an energy-effective way. Zhou and Zhang [26] proposed compensating tasks with a higher delay-to-deadline ratio to minimize the maximal ratio among users, which was solved using an evolutionary algorithm. Hejja et al. [27] maximized the number of serviced offloading requests under the network function virtualization framework. Finally, Meng et al. [28] developed a task dispatching and scheduling algorithm that focused more on computing node selection and task scheduling to maximize the number of tasks meeting deadline requirements. We summarize part of the studies mentioned above in Table 1.

Table 1. Summary of the discussed work.

Work	Nodes with Computing Power	Variables to Be Optimized	Objective	Methodology
[7]	Edge Server (ES), Cloud Server (CS)	$\boldsymbol{\lambda}$ [5], $\boldsymbol{x}$ [1], $\boldsymbol{b}$ [2], $\boldsymbol{\alpha}$ [3]	DCM	Decomposition and KKT Conditions
[12]	UD, ES, CS	$\boldsymbol{x}$, $\boldsymbol{y}$ [4], $\boldsymbol{\alpha}$	DCM	ADMM
[13]	ES, CS	$\boldsymbol{y}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	DCM	Actor-Critic based DRL
[29]	UD, ES	$\boldsymbol{\lambda}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	DCM	The Lagrange multiplier method
[18]	UD, ES	$\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\alpha}$	ECM	DDPG
[30]	UD, ES	$\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	Computation Rate Maximization	Lyapunov Optimization and DRL
[26]	ES	$\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	Minimize Maximal Delay ratio	Evolutionary Algorithm
[19]	UD, ES	$\boldsymbol{x}$, $\boldsymbol{\alpha}$	ECM	Ant Colony-based algorithm
[24]	UD, ES	$\boldsymbol{x}$, $\boldsymbol{c}$ [6]	DCM and Revenue Maximization	Stackelberg Game
[25]	UD, ES	$\boldsymbol{x}$, $\boldsymbol{c}$	ECM and Revenue Maximization	Market Auction Theory
Our Work	ES	$\boldsymbol{x}$, $\boldsymbol{b}$, $\boldsymbol{\alpha}$	Acceptance Ratio Maximization	WPR

[1]: $\boldsymbol{x}$ is the binary offloading vector; [2]: $\boldsymbol{b}$ is the bandwidth allocation vector; [3]: $\boldsymbol{\alpha}$ is the computing resource allocation vector; [4]: $\boldsymbol{y}$ denotes the computation node selection vector; [5]: $\boldsymbol{\lambda}$ denotes the splitting ratio; [6]: $\boldsymbol{c}$ denotes the pricing vector.

Furthermore, recent research has explored combining deep reinforcement learning with computation offloading to enhance performance [13,17,30–33]. However, these approaches do not address the problem of maximizing the number of accepted requests among a flood of requests with limited resources in an edge-enabled asymmetrical network. This requires a system of efficient allocation of resources and selection of the appropriate requests while considering their respective delay requirements. This paper proposes a solution to solve this problem, wherein tasks are processed at the edge server, and rejected tasks are considered to be task failures. Notably, our proposed algorithm can function as a supplementary approach and as a standalone approach.

3. System Model

As shown in Figure 1, the system consists of a single BS and M UDs. The BS is equipped with an edge computing server, having a computing power of Fe (in CPU cycles per second). The BS is connected to the edge server via a fiber link [34] to provide computing services for UDs. UDs issue computation offloading requests to the associated BS if local computing resources cannot complete the task within the deadline T^{dl}, and such requests are considered to be timeouts if they are rejected. It is assumed that the computation-intensive tasks arrive at the beginning of a schedule interval [35] and only one task is generated from each UD. Local processing is not considered and requests with no resources allocated are treated as timeouts.

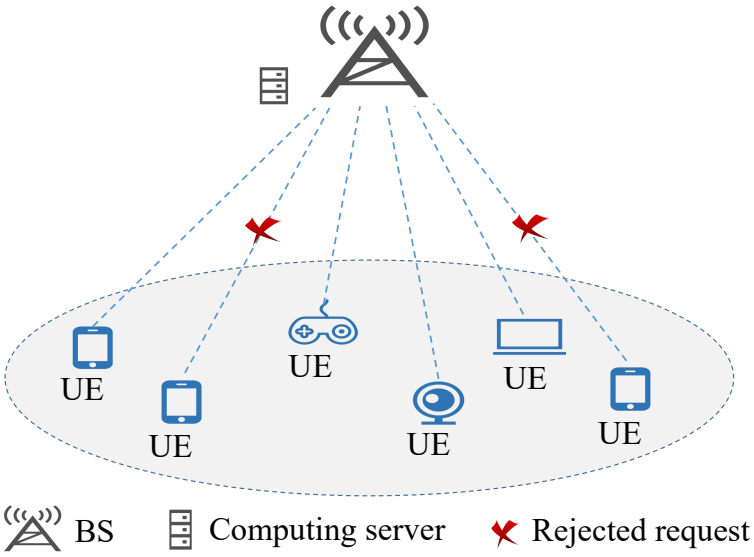


Figure 1. System model.

There are K kinds of computation applications running in the system. In this paper, it is assumed that the input of computational application k is one of the F^k input files. A computation request of application k is specified by the task input size l_f^k (in bits), the computation load L_f^k (the desired computing resource, in CPU cycles), and the task deadline T_f^k (this can also be a QoS-related delay parameter). Thus, a UD m requesting computation offloading of application k with task input f ($f \in \mathcal{F}^k = \{1, 2, 3, \cdots, F^k\}$) can be specified with $r_m = (l_f^k, L_f^k, T_f^k)$. r_m indicates the computation task from UD m in the sequel. For simplicity of notations, the parameters of task r_m are written as (l_m, L_m, T_m^{dl}).

When UDs cannot complete their tasks within the given deadline, they request the associated BS for computation offloading [3]. While some previous research considered local processing power on the UDs, this paper excludes UDs that can process tasks without exceeding the deadline ($L_m / f_m \leq T_m^{dl}$). As a result, the BS can either accept or decline the computation offloading requests, depending on its processing capabilities. The BS tries to accommodate the requests to the best of its abilities. When task r_m is accepted (i.e., $x_m = 1$), the BS ensures that resource allocation meets the delay requirement (or QoS-related delay requirement). If task r_m is denied (i.e., $x_m = 0$), the BS incurs a penalty. The BS makes decisions on joint offloading request acceptance and resource allocation with the aim of maximizing the number of accepted requests.

3.1. Transmission Model

The whole system is presumed to function in an orthogonal frequency division multiple access (OFDMA) mode with a bandwidth of B (in hertz). In this way, interference is not considered in our model. Noticing the asymmetry in uplink and downlink, the result download stage is not considered in our model [36]. Wireless access bandwidth is only allocated to accepted requests and the BS assigns a portion bandwidth α_m to UD m to upload the task input file (of size l_m). Consequently, the maximal upload data rate between UD m and the BS can be expressed as follows:

$$R_m = \alpha_m B \log_2 (1 + \frac{p_m h_m^2}{\sigma^2}) = \alpha_m \hat{R}_m, \tag{1}$$

where p_m is the upload transmission power (usually determined by the association control scheme) of UD m. h_m is the instantaneous channel gain between UD m and the BS. As the transmission may occur over several slots, the averaged channel gain $\overline{h}_m$ is employed to substitute the instantaneous channel gain [7] across multiple frames (with channel

estimation technologies [37]). Supposing request r_m from UD m is accepted, the averaged upload transmission time for UD m to upload its computation task input can be written as:

$$t_m^{up} = \frac{l_m}{\alpha_m B \log_2(1 + \frac{p_m \overline{h}_m^2}{\sigma^2})} = \frac{l_m}{\alpha_m \overline{R}_m}, \tag{2}$$

where $\overline{R}_m$ is the average of $B\log_2(1 + \frac{p_m \overline{h}_m^2}{\sigma^2})$ during t_m^{up}.

In addition, the allocated access bandwidth of the BS cannot exceed its capacity:

$$\sum_{m \in \mathcal{M}_1} \alpha_m \leq 1, \tag{3}$$

where $\mathcal{M}_1$ denotes the set of accepted requests from UDs.

3.2. Computing Model

The heterogeneity in computing resource requirements among accepted requests necessitates efficient allocation of computing resources by the edge server. $\mathcal{M}_1$ denotes the set of accepted requests and $\mathcal{M}_0$ denotes the set of rejected requests. The BS assigns a fraction β_m of its computing resources to UD m to process the computation task. The resulting computation delay can be written as:

$$t_m^e = \frac{L_m}{\beta_m Fe}. \tag{4}$$

The allocation of computing resources must adhere to its capacity constraint, which can be formally expressed as:

$$\sum_{m \in \mathcal{M}_1} \beta_m \leq 1. \tag{5}$$

3.3. Problem Formulation

If a request r_m is accepted, the BS ensures that the associated delay constraint is not violated. This requirement can be expressed as:

$$T_m = t_m^{up} + t_m^e \leq T_m^{dl}, \forall m \in \mathcal{M}_1. \tag{6}$$

The main objective is to maximize the number of accommodated requests subject to the constraints imposed by the limited system resources and deadlines, which can be formally expressed as:

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta}} : & \sum_{m=1}^{M} \mathbb{1}(x_m) \\ \text{s.t.} \quad \text{C1} : & \sum_{m=1}^{M} x_m \alpha_m \leq 1 \\ \text{C2} : & \sum_{m=1}^{M} x_m \beta_m \leq 1 \\ \text{C3} : & \; x_m \left(\frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe}\right) \leq T_m^{dl}, \forall m \in \mathcal{M}. \end{aligned} \tag{P1}$$

$\mathbb{1}(x_m)$ is an indicator function and takes value one when $x_m = 1$ and zero otherwise. Constraints $C1$ and $C2$ ensure that the bandwidth and computing resources allocated to the accepted requests should not exceed the system's capacity. Constraint $C3$ ensures that the delay requirements of the accepted requests are met.

Problem (P1) is known to be intractable, due to the non-smooth and non-convex nature of the objective function. To overcome this challenge, Problem (P1) is transformed into

Problem (P2) with the objective of minimizing the total delay cost of all requests. This transformation reduces the search domain of Problem (P1) and enables more efficient solution approaches to solve Problem (P2). Two key observations form the basis of this transformation. First, for an accepted request, the processing delay is always less than the corresponding deadline, while a rejected request receives a relatively large penalty. In this way, the system delay cost can be reduced by accepting more requests. Second, the optimal allocation scheme to Problem (P1) does not necessarily minimize the overall delay cost. Such a scheme could lead to higher delays compared to Problem (P2), which leaves more vacant resources for other requests. Consequently, solutions to Problem (P1) can be obtained by first solving the delay cost minimization (P2) and pushing the resource allocation scheme near to the deadline.

In this paper, rejected requests are treated as timed-out and discarded, incurring penalties. Specifically, the penalty of a timed-out request is denoted as $\eta T_n^{dl}, \forall n \in \mathcal{M}_0$, where $\eta \gg 1$ is a constant factor for all requests. This penalty factor reflects the severity of a timed-out request in terms of delay cost. To account for these penalties, Problem (P2) is reformulated by adding penalty terms for timed-out requests:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\beta}} : & \sum_{m=1}^{M} x_m(t_m^{up} + t_m^e) + (1 - x_m)\eta T_m^{dl} \\ \text{s.t.} \quad & \text{C1,C2,C3.} \end{aligned} \tag{P2}$$

Problem (P2) is a challenging MINLP problem with coupled decision variables. To address this challenge, we follow previous works [38] and Problem (P2) and decompose the problem into two sub-problems: request acceptance and resource allocation. The binary request acceptance problem can be solved with a coordinate descent algorithm [39]. Note that both request acceptance and resource allocation influence the final effect. Specifically, this paper focuses on solving the resource allocation sub-problem to gain insight into Problem (P2).

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\beta}} : & \sum_{m\in\mathcal{M}_1} (t_m^{up} + t_m^e) + \sum_{n\in\mathcal{M}_0} \eta T_n^{dl} \\ \text{s.t.} \quad \text{C4} : & \sum_{m\in\mathcal{M}_1} \alpha_m \leq 1 \\ \text{C5} : & \sum_{m\in\mathcal{M}_1} \beta_m \leq 1 \\ \text{C6} : & \frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe} \leq T_m^{dl}, \forall m \in \mathcal{M}_1. \end{aligned} \tag{P2.1}$$

It can be seen from above that, once $\boldsymbol{x}$ is given ($\mathcal{M}_0$ and $\mathcal{M}_1$ are determined), Problem (P2.1) can be reformulated as a convex optimization problem [38]. Once an optimal resource allocation scheme is obtained, we readjust the resource allocation result in $\mathcal{M}_1$ and try to allocate resources to request in $\mathcal{M}_0$, to accommodate more requests without violating the deadline constraints, and, then, obtain the solutions to the original Problem (P1). This process allows us to iteratively refine our solutions and obtain an optimized allocation of system resources that maximizes the number of accommodated requests within the constraints of the system's limited resources and deadlines.

4. The Water-Pumping Algorithm

In this section, the classic Lagrange multiplier method and KKT conditions are adopted to derive the solution to Problem (P2.1). Based on this solution, the water pumping and refilling algorithm is proposedto solve the initial asymmetric resource allocation Problem, as stated in Problem (P1).

4.1. The Delay Minimum Solution

By introducing Lagrange multipliers $\lambda_1, \lambda_2, \boldsymbol{\nu} = (\nu_1, \nu_2, \cdots, \nu_m), \forall m \in \mathcal{M}_1$, the Lagrange function of (P2.1) can be formulated as:

$$L(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda_1, \lambda_2, \boldsymbol{\nu}) = \sum_{m \in \mathcal{M}_1} (\frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe}) + \sum_{n \in \mathcal{M}_0} \eta T_m^{dl} + \lambda_1 (\sum_{m \in \mathcal{M}_1} \alpha_m - 1) + \lambda_2 (\sum_{m \in \mathcal{M}_1} \beta_m - 1) + \sum_{m \in \mathcal{M}_1} \nu_m (\frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe} - T_m^{dl}). \quad (7)$$

Based on Equation (7) and in-depth analysis of Problem (P2.1), using KKT conditions, the following corollaries can be obtained.

Corollary 1. *The necessary condition for an optimal bandwidth and computing resource allocation scheme for $\mathcal{M}_1$ is given by:*

$$(\alpha_m, \beta_m) = (\frac{\sqrt{\frac{(1+\nu_m) l_m}{\overline{R}_m}}}{\sqrt{\lambda_1}}, \frac{\sqrt{\frac{(1+\nu_m) L_m}{Fe}}}{\sqrt{\lambda_2}}) = (\frac{\sqrt{\frac{(1+\nu_m) l_m}{\overline{R}_m}}}{\sum_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) l_i}{\overline{R}_i}}}, \frac{\sqrt{\frac{(1+\nu_m) L_m}{Fe}}}{\sum_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) L_i}{Fe}}}). \quad (8)$$

Proof. Please see the detailed proof in Appendix A. □

The optimal bandwidth and computing resource allocation scheme for UD m can be obtained from Equation (8). It can be observed that the optimal bandwidth allocation is proportional to $\sqrt{\frac{l_m}{\overline{R}_m}}$, which implies that requests with better channel conditions and larger input data size receive a larger share of the bandwidth allocation. Similarly, requests with higher computation loads are allocated more edge computing resources.

Corollary 2. *The optimal allocation scheme achieves the minimal delay cost for all $m \in \mathcal{M}_1$ when the Lagrange multiplier ν_m takes the same value (e.g., 0), which can be denoted as:*

$$(\alpha_m^*, \beta_m^*) = (\frac{\sqrt{\frac{l_m}{\overline{R}_m}}}{\sum_{i \in \mathcal{M}_1} \sqrt{\frac{l_i}{\overline{R}_i}}}, \frac{\sqrt{\frac{L_m}{Fe}}}{\sum_{i \in \mathcal{M}_1} \sqrt{\frac{L_i}{Fe}}}). \quad (9)$$

Proof. Please see the detailed proof in Appendix B. □

Equation (9) indicates that the delay-minimum resource allocation scheme to Problem (P2.1) in this context leads to the exhaustion of all system resources for accepted requests, without taking into account the possibility of over-provisioning a request, based on its maximum acceptable delay. In general, the ratio of the user's processing delay to its deadline under the delay-minimum scheme is less than one ($T_m < T_m^{dl}, \forall m \in \mathcal{M}_1$), where the processing delay of an accepted request r_m can be represented as:

$$T_m = \frac{\sqrt{\lambda_1}}{\sqrt{1+\nu_m}} \sqrt{\frac{l_m}{\overline{R}_m}} + \frac{\sqrt{\lambda_2}}{\sqrt{1+\nu_m}} \sqrt{\frac{L_m}{Fe}}. \quad (10)$$

Upon comparing Equation (8) with Equation (9), it can be observed that the Lagrange multipliers $\nu_m, \forall m \in \mathcal{M}_1$ play a crucial role in regulating the allocation of resources, which, in turn, impacts the processing delays of accepted requests. Thus, the proposed WPR algorithm increases the number of accepted requests by reallocating resources (setting ν_m) to extend the processing delays of accepted requests up to their respective deadlines ($T_m \approx T_m^{dl}, \forall m \in \mathcal{M}_1$).

4.2. Water Pumping

It can be inferred from Equation (10) that T_m can be extended to be equal to T_m^{dl} by adjusting ν_m, while keeping λ_1, λ_2 (which can be calculated with $\boldsymbol{\nu} = (\nu_1, \nu_2, \cdots, \nu_m), \forall m \in \mathcal{M}_1$ using Equation (8)) unchanged. This can be denoted as:

$$T_m^{dl} = \frac{\sqrt{\lambda_1}}{\sqrt{1+\nu_m^p}}\sqrt{\frac{l_m}{\overline{R}_m}} + \frac{\sqrt{\lambda_2}}{\sqrt{1+\nu_m^p}}\sqrt{\frac{L_m}{Fe}}. \tag{11}$$

Based on this observation, we propose the term "*water pumping* " to describe the procedure whereby the value of ν_m, the water level parameter of an accepted request r_m, is adjusted from its initial value ν_m to a new value ν_m^p. To further characterize this process, we define the ratio a_m as $a_m = \frac{T_m}{T_m^{dl}} = \frac{\sqrt{1+\nu_m^p}}{\sqrt{1+\nu_m}}$. By manipulating the value of ν_m, the processing delay T_m is extended to the desired maximal acceptable deadline T_m^{dl} for request r_m:

$$\nu_m^p = a_m^2(1+\nu_m) - 1. \tag{12}$$

Notably, the proposed scheme can be readily extended to a cloud-edge collaboration model [12,40,41]. In this case, the ratio a_m can be redefined as $\frac{T_m}{T_m^{dl} - T_{rtt}}$, where T_{rtt} denotes the round-trip time between the edge node and the remote cloud server. As we adjust the value of ν_m to ν_m^p, the vector $\boldsymbol{\nu} = (\nu_1, \nu_2, \cdots, \nu_m), \forall m \in \mathcal{M}_1$ shifts to the new value $\boldsymbol{\nu}^p = (\nu_1, \nu_2, \cdots, \nu_m^p), \forall m \in \mathcal{M}_1$. λ_1 and λ_2 shifts to λ_1^p and λ_2^p, accordingly. The decrement of $\sqrt{\lambda_1}, \sqrt{\lambda_2}$, which is termed the "*pumped water* ", can be expressed as follows:

$$\Delta_1 = \sqrt{\lambda_1} - \sqrt{\lambda_1^p} = \sqrt{\frac{l_m}{\overline{R}_m}}\left(\sqrt{1+\nu_m} - \sqrt{1+\nu_m^p}\right) = (1-a_m)\sqrt{1+\nu_m}\sqrt{\frac{l_m}{\overline{R}_m}}, \tag{13}$$

$$\Delta_2 = \sqrt{\lambda_2} - \sqrt{\lambda_2^p} = \sqrt{\frac{L_m}{Fe}}\left(\sqrt{1+\nu_m} - \sqrt{1+\nu_m^p}\right) = (1-a_m)\sqrt{1+\nu_m}\sqrt{\frac{L_m}{Fe}}. \tag{14}$$

4.3. Water Refilling

To accommodate request r_{m+1}, the pumped water Δ_1 and Δ_2 should be refilled to ensure that r_{m+1} does not violate its deadline constraint. The procedure of setting a feasible ν_{m+1} for r_{m+1} is termed "*water refilling*". Successful water refilling involves finding a suitable r_{m+1} and setting ν_{m+1}, within the following constraints:

$$\sqrt{\lambda_1'} = \sqrt{1+\nu_1}\sqrt{\frac{l_1}{\overline{R}_1}} + \cdots + \sqrt{1+\nu_{m-1}}\sqrt{\frac{l_{m-1}}{\overline{R}_{m-1}}} + \sqrt{1+\nu_m^p}\sqrt{\frac{l_m}{\overline{R}_m}} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} = \sqrt{\lambda_1^p} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} \le \sqrt{\lambda_1}, \tag{15}$$

$$\sqrt{\lambda_2'} = \sqrt{1+\nu_1}\sqrt{\frac{L_1}{Fe}} + \cdots + \sqrt{1+\nu_{m-1}}\sqrt{\frac{L_{m-1}}{Fe}} + \sqrt{1+\nu_m^p}\sqrt{\frac{L_m}{Fe}} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{L_{m+1}}{Fe}} = \sqrt{\lambda_2^p} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{L_{m+1}}{Fe}} \le \sqrt{\lambda_2}, \tag{16}$$

$$T_{m+1} = \frac{\sqrt{\lambda_1'}}{\sqrt{1+\nu_{m+1}}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} + \frac{\sqrt{\lambda_2'}}{\sqrt{1+\nu_{m+1}}}\sqrt{\frac{L_{m+1}}{Fe}} \le T_{m+1}^{dl}. \tag{17}$$

Here $\sqrt{\lambda_1'}, \sqrt{\lambda_2'}$ are obtained with $\boldsymbol{\nu}' = (\nu_1, \nu_2, \cdots, \nu_{m-1}, \nu_m', \nu_{m+1}), \forall i \in \mathcal{M}'_1 \mathcal{M}'_1 = \mathcal{M}_1 \cup \{m+1\}$ according to Equations (A14) and (A15) ($\sqrt{\lambda_1^p}$ and $\sqrt{\lambda_2^p}$ are obtained with $\boldsymbol{\nu}^p$). The reason for taking the inequality in Equations (15) and (16) is to ensure there is no violation of the resource constraints C1 and C2. Equation (17) guarantees that the

allocated resource for r_{m+1} meets its deadline requirement. In this regard, compared with Equations (13) and (14), Equations (15) and (16) can be rewritten as:

$$\sqrt{1+\nu_{m+1}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} \leq \Delta_1, \tag{18}$$

$$\sqrt{1+\nu_{m+1}}\sqrt{\frac{L_{m+1}}{Fe}} \leq \Delta_2. \tag{19}$$

Furthermore, the value of ν_{m+1} for the newly accepted request r_{m+1} can be decided with the following formula:

$$\nu_{m+1} = min\left\{\frac{\overline{R}_{m+1}}{l_{m+1}}(\Delta_1)^2 - 1, \frac{Fe}{L_{m+1}}(\Delta_2)^2 - 1\right\}, T_{m+1} \leq T^{dl}_{m+1}. \tag{20}$$

Unfortunately, a single trial of "water pumping" may not necessarily result in successful "water refilling", so situations where multiple trials of "water pumping" are necessary to ensure successful refilling should be considered. Denote $\mathcal{S}$ ($\mathcal{S} \subset \mathcal{M}_1$) as the current set of "pumped requests", containing the requests having shrunken ν ($\nu_i \to \nu_i^p \quad \forall i \in \mathcal{S}$), and $\mathcal{U}$ ($\mathcal{U} \subset \mathcal{M}_1, \mathcal{U} \bigcap \mathcal{S} = \emptyset$ and $\mathcal{U} \bigcup \mathcal{S} = \mathcal{M}_1$) as the set of "unpumped requests". According to a predetermined rule, (the pumping policy $\mathbb{P}$), a request can be selected from $\mathcal{U}$ to perform "water pumping". Assuming r_{m+1} is the chosen request to be "refilled", based on another predetermined rule, the refilling policy $\mathbb{R}$, after several pumping trials, ν_{m+1} can be determined without violating the following constraints:

$$\sqrt{\lambda_1^{'}} = \sum_{i\in\mathcal{S}}\sqrt{1+\nu_i^p}\sqrt{\frac{l_i}{\overline{R}_i}} + \sum_{j\in\mathcal{U}}\sqrt{1+\nu_j}\sqrt{\frac{l_j}{\overline{R}_j}} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} \leq \sqrt{\lambda_1}, \tag{21}$$

$$\sqrt{\lambda_2^{'}} = \sum_{i\in\mathcal{S}}\sqrt{1+\nu_i^p}\sqrt{\frac{L_i}{Fe}} + \sum_{j\in\mathcal{U}}\sqrt{1+\nu_j}\sqrt{\frac{L_j}{Fe}} + \sqrt{1+\nu_{m+1}}\sqrt{\frac{L_{m+1}}{Fe}} \leq \sqrt{\lambda_2}, \tag{22}$$

Compared with Equations (18) and (19), Equations (21) and (22) can be rewritten as follows:

$$\Sigma\Delta_1^{'} = \sqrt{\lambda_1} - \sqrt{\lambda_1^{'}} = \sum_{i\in\mathcal{S}}(1-a_i)\sqrt{1+\nu_i}\sqrt{\frac{l_i}{\overline{R}_i}} - \sqrt{1+\nu_{m+1}}\sqrt{\frac{l_{m+1}}{\overline{R}_{m+1}}} \geq 0, \tag{23}$$

$$\Sigma\Delta_2^{'} = \sqrt{\lambda_2} - \sqrt{\lambda_2^{'}} = \sum_{i\in\mathcal{S}}(1-a_i)\sqrt{1+\nu_i}\sqrt{\frac{L_i}{Fe}} - \sqrt{1+\nu_{m+1}}\sqrt{\frac{L_{m+1}}{Fe}} \geq 0. \tag{24}$$

In this way, the ν_{m+1}, after multiple pumping trials of can be obtained from:

$$\nu_{m+1} = min\left\{\frac{\overline{R}_{m+1}}{l_{m+1}}(\Sigma\Delta_1^{'})^2 - 1, \frac{Fe}{L_{m+1}}(\Sigma\Delta_2{'})^2 - 1\right\}, \tag{25}$$

It should be noted that requests $r_i, \forall i \in \mathcal{S}$ do not violate the deadline requirements after accepting the new request r_{m+1}. Equation (11) indicates that the processing time of a pumped request is extended to its deadline. Equations (15) and (21) guarantee that $\sqrt{\lambda_1^{'}} \leq \sqrt{\lambda_1}$. Similarly, $\sqrt{\lambda_2^{'}} \leq \sqrt{\lambda_2}$ holds. As a result, the processing delay of a pumped request can be reformulated as:

$$T_m^{'} = \frac{\sqrt{\lambda_1^{'}}}{\sqrt{1+\nu_m^{'}}}\sqrt{\frac{l_m}{R_m}} + \frac{\sqrt{\lambda_2^{'}}}{\sqrt{1+\nu_m^{'}}}\sqrt{\frac{L_m}{Fe}} \leq \frac{\sqrt{\lambda_1}}{\sqrt{1+\nu_m^{'}}}\sqrt{\frac{l_m}{R_m}} + \frac{\sqrt{\lambda_2}}{\sqrt{1+\nu_m^{'}}}\sqrt{\frac{L_m}{Fe}} = T_m^{dl}. \quad (26)$$

The WPR is all about how to allocate network resources to provide near-to-deadline services for accepted computation offloading requests. The algorithm converges to its final result by continuously pumping and refilling until no request can be successfully added. The whole procedure of the WPR algorithm is summarized in Algorithm 1, and some notations are listed in Table 2.

Table 2. Summary of notations used in the WPR algorithm.

Notation	Description
$\mathbb{P}$	the pumping policy deciding the pumping order of the accepted requests
$\mathbb{R}$	the refilling policy deciding the refilling order of the rejected requests
α_m	the bandwidth fraction allocated to request r_m
β_m	the computing resource fraction allocated to request r_m
$\mathcal{M}_1, \mathcal{M}_1^c, \mathcal{M}_0$	the set of accepted requests, current accepted requests, and rejected requests
$\mathcal{M}_1^l$	the latest set after the last successful refilling
$\mathcal{U}$	the set of "unpumped requests" $\mathcal{U} \subset \mathcal{M}_1, \mathcal{U} \bigcap \mathcal{S} = \emptyset$ and $\mathcal{U} \bigcup \mathcal{S} = \mathcal{M}_1$
$\mathcal{S}$	$\mathcal{S} \subset \mathcal{M}_1$ the set of "pumped requests", $\nu_i \rightarrow \nu_i^p, \forall i \in \mathcal{S}$ according to Equation (11)
$\lambda_1, \lambda_1^p, \lambda_1^{'}$	the Lagrange multipliers obtained with $\boldsymbol{\nu}, \boldsymbol{\nu^p}, \boldsymbol{\nu}^{'}$ according to Equation (8)
$\boldsymbol{\nu}$	the Lagrange multipliers of accepted requests $\boldsymbol{\nu} = (\nu_1, \nu_2, \cdots, \nu_m), \forall m \in \mathcal{M}_1$
$\boldsymbol{\nu^p}$	the Lagrange multipliers of accepted requests $\boldsymbol{\nu^p} = (\nu_1, \nu_2, \cdots, \nu_m^p), \forall m \in \mathcal{M}_1$
$\Sigma\Delta_1^{'}, \Sigma\Delta_2^{'}$	the cumulative "pumped water" after multiple pumping trials

4.4. Complexity Analysis

The proposed WPR algorithm provides a solution to Problem (P1) and offers flexibility in designing service strategies by allowing the choice of requests to shrink during each iteration. It is important to note that the WPR algorithm converges within at most M refilling iterations, provided that the exit condition in Line 14 of Algorithm 1 is not met. Assuming that $|\mathcal{M}_1^c| = m_1$, the last successful one-to-one pumping and refilling happens at this point in time. After this moment, multiple rounds of pumping are necessary to ensure a successful refilling. Once $|\mathcal{M}_1^c| = m_2$, no further refilling attempts will succeed, which means the end of the algorithm. Therefore, the total pumping procedure will take $m_1 + (m_1 + 1) + (m_1 + 2) + \cdots + (m_2 - 1) + m_2 \leq |\mathcal{M}|^2$ iterations. Thus, the time complexity of the WPR algorithm is $\mathcal{O}(M^2)$.

Algorithm 1 : Water Pumping and Refilling

Input: initial accepted requests set $\mathcal{M}_1$, initial $\boldsymbol{\nu}$ of $\mathcal{M}_1$, pumping policy $\mathbb{P}$ and refilling policy $\mathbb{R}$;

Output: final accepted requests set $\mathcal{M}_1^f$, final $\boldsymbol{\nu}$ of $\mathcal{M}_1$;

current set of accepted requests that have not been pumped $\mathcal{U} = \mathcal{M}_1$;
current shrunk set $\mathcal{S} = \emptyset$, last $\boldsymbol{\nu}^l = \boldsymbol{\nu}$ of the accepted requests $\mathcal{M}_1$ after a successful refilling;
current rejected requests set $\mathcal{M}_0 = \mathcal{M} - \mathcal{M}_1^c$, current $\boldsymbol{\nu}^c$ of current accepted requests $\mathcal{M}_1^c$;
last $\mathcal{M}_1^l = \mathcal{M}_1^c$ after reset (used as the condition to exit the loop) $\Sigma\Delta_1 = 0, \Sigma\Delta_2 = 0$;
while $\mathcal{U} \neq \emptyset$ **do**:
    get the request r_m to be pumped from $\mathcal{U}$ according to $\mathbb{P}$, $\mathcal{U} = \mathcal{U} - \{m\}$;
    get a_m and update $\nu_m^{'}$ with Equation (12) (the water pumping), update $\boldsymbol{\nu}^c$;
    get Δ_1, Δ_2 with Equations (13) and (14), update $\Sigma\Delta_1 += \Delta_1, \Sigma\Delta_2 += \Delta_2$;
    get the request r_{m+1} to be refilled from $\mathcal{M}_0$, according to $\mathbb{R}$, obtain ν_{m+1} with Equation (25) (water refilling), and check whether r_{m+1} exceeds its deadline;
    if the water refilling succeeds in accommodating r_{m+1} **then**
        $\mathcal{M}_1^c = \mathcal{M}_1^c \cup \{m+1\}, \boldsymbol{\nu}^c = \boldsymbol{\nu}^c \cup \{\nu_{m+1}\}, \boldsymbol{\nu}^l = \boldsymbol{\nu}^c, \mathcal{U} = \mathcal{U} \cup \{m+1\}, \mathcal{M}_0 = \mathcal{M}_0 - \{m+1\}, \Sigma\Delta_1 = 0, \Sigma\Delta_2 = 0$;
    else
        $\mathcal{M}_1^c = \mathcal{M}_1^c, \boldsymbol{\nu}^c = \boldsymbol{\nu}^c, \mathcal{M}_0 = \mathcal{M}_0$;
        if $\mathcal{U} = \emptyset$ **then** ▷ reset $\mathcal{U}$ until no new request is accepted
            if $\mathcal{M}_1^c \neq \mathcal{M}_1^l$ **then**
                $\mathcal{M}_1^l = \mathcal{M}_1^c$;
                $\mathcal{U} = \mathcal{M}_1^c, \Sigma\Delta_1 = 0, \Sigma\Delta_2 = 0$;
            else
                return scheme $\mathcal{M}_1^c, \boldsymbol{\nu}^l$.
return scheme $\mathcal{M}_1^c, \boldsymbol{\nu}^l$.

5. Simulation Results and Discussion

In this section, we present results and discussions concerning the edge-enabled asymmetrical network under different parameters.

5.1. Simulation Setting

The system parameters are defined as follows. The bandwidth of the BS is 8×10^6 MHz, and the computing capacity of the connected edge server is $Fe = 1 \times 10^{10}$ CPU cycles/second. The channel model utilized in this study was the same as the one presented in [38]. UDs upload task data with a fixed transmission power $p_m = 0.2 \quad \forall m \in \mathcal{M}$. The system contains $M = 20$ UDs, and it considers only one type of computation application, denoted by $K = 1$. Accordingly, $F = 10$ input files are considered, with the input size l_m^f (in bits) ranging from $\left[l_{min} = 1 \times 10^6, l_{max} = 20 \times 10^6\right]$. Specifically, the input size of r_m with input file f takes the value of $l_m^f = l_{min} + (f-1)\frac{l_{max} - l_{min}}{F}$ following uniform distribution, Zipf distribution [41,42] (skewness factor $\alpha = 1$) prioritizing small loads and Zipf distribution prioritizing large loads. The computation load (L_m) of each task ranges from $\left[L_{min} = 0.5 \times 10^8, L_{max} = 4 \times 10^8\right]$ (in CPU cycles) and computation load of r_m with input file f takes a value of $L_m^f = L_{min} + (f-1)\frac{L_{max} - L_{min}}{F}$ following the same distribution as l_m^f. $T_m^{dl} = 0.4$ and the penalty factor $\eta = 10$. Unless otherwise specified, the results were obtained based on uniform distribution. In this paper, the default policy $\mathbb{P}$ selects the request r_m with the highest a_m in $\mathcal{U}$ to pump first and $\mathbb{R}$ tries to refill the request r_i with the smallest $\sqrt{\lambda_1}\sqrt{\frac{l_i}{R_i}} + \sqrt{\lambda_2}\sqrt{\frac{L_i}{Fe}}$ in $\mathcal{M}_0$. It should be noted that other $\mathbb{P}$ and $\mathbb{R}$ can be customized and adopted, such as smallest file size first (SFWPR), and best channel

condition first (which is not considered in this paper). The following baseline algorithms were used in this paper:

- Delay cost minimization (DCM): this scheme allocates resources for accepted requests with the aim of minimizing system delay costs. In cases where a request is rejected, the DCM scheme imposes a penalty instead of the processing delay.
- Water pumping and refilling, basing on DCM (WPDCM): this scheme uses results obtained from DCM as the input $\mathcal{M}_1$ of WPR and sets each item of ν to 1.
- Water pumping and refilling (WPR): The initial $\mathcal{M}_1$ only includes the request with the smallest $\sqrt{\lambda_1}\sqrt{\frac{l_i}{R_i}} + \sqrt{\lambda_2}\sqrt{\frac{L_i}{Fe}}$ which is the refilling policy $\mathbb{R}$ used by default.
- Smallest input file first water pumping and refilling (SFWPR): using the default policy $\mathbb{P}$ while refilling requests with the smallest input size. The initial $\mathcal{M}_1$ only includes the request with the smallest input size.

5.2. Result Discussion

Figure 2 illustrates the relationship between the number of UDs and the acceptance ratio of offloading requests in the system. As the number of UDs requesting computation offloading increased, the acceptance ratio of requests tended to decrease. The reason for this is clear, the limited system resources could not deal with requests beyond the system's capabilty. The WPR-based algorithms achieved a higher acceptance ratio than the DCM algorithm. The reason for this is that the DCM algorithm prioritized minimization of the sum processing delay of the accepted requests, which came at the expense of reducing the system's capacity to handle a high volume of user requests. In other words, while minimizing processing delay is a desirable goal to improve system performance, it may result in reduced capacity to handle additional user requests. Moreover, the WPR scheme proposed in this study demonstrated competitive performance, and the highest acceptance ratio among all the schemes considered, including WPDCM. Therefore, the proposed WPR algorithm can be used as a supplementary algorithm to DCM and as an independent algorithm, offering competitive results.

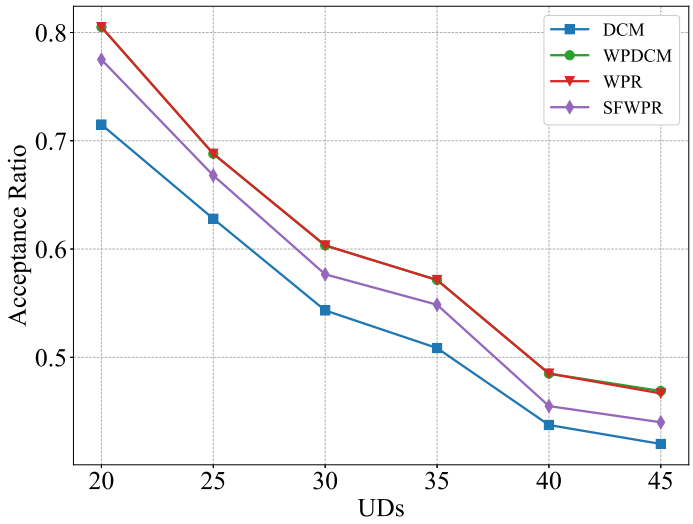
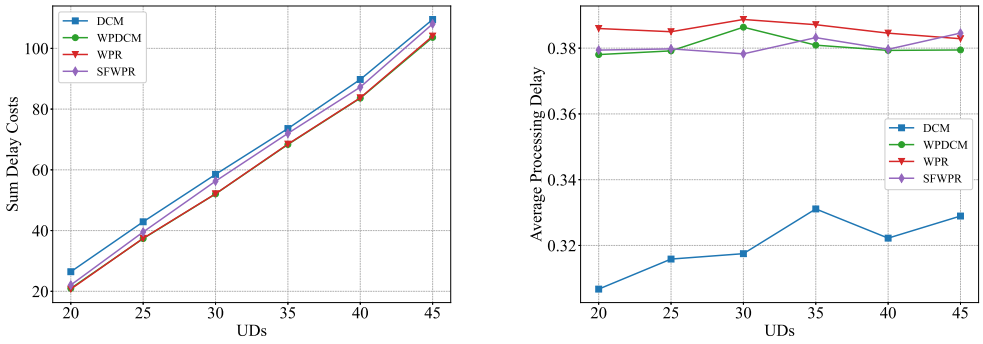


Figure 2. Acceptance ratio versus UDs.

Figure 3a illustrates that the aggregate system delay cost increased proportionally with the number of offloading user devices (UDs). This escalation is primarily attributed to the penalty incurred by rejecting redundant offloading requests that exceed the system's resource capacity. Notably, the DCM algorithm incurred the highest delay cost, mainly due to its low acceptance ratio. In contrast, Figure 3b illustrates that the WPR-based algorithm achieved a higher acceptance ratio than the DCM algorithm. The DCM algorithm aimed to

minimize the processing delay of each accepted request, while the WPR-based algorithm permitted the processing delay of an accepted request to approximate its deadline. This approach left more system resources available to accommodate additional requests.



(a) **(b)**

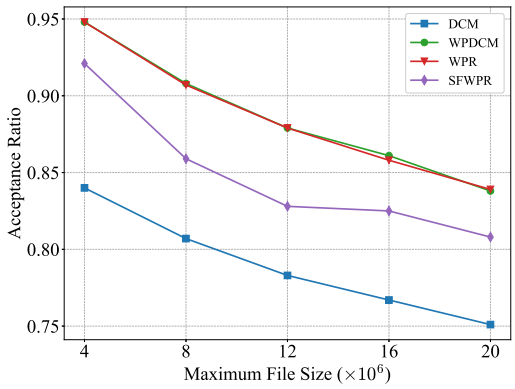
Figure 3. Delay cost versus UDs: (**a**) sum delay cost of all UDs; (**b**) average processing delay for an accepted request.

From Figure 4, several conclusions can be drawn. Firstly, it can be observed that all schemes achieved higher acceptance ratios when the input size of the majority of tasks had a small range, as depicted in Figure 4a,b. This is due to the fact that smaller service loads require fewer resources. Secondly, the proposed SFWPR scheme outperforms the DCM scheme in maximizing the acceptance ratio When more tasks carried smaller input sizes. As shown in Figure 4b, the acceptance ratio of SFWPR was higher than that of DCM and was very close to those of WPR and WPDCM. However, it should be noted that SFWPR might not be suitable for scenarios wherein tasks with large input sizes constitute the majority. Lastly, as the input size of tasks increased, the acceptance ratio decreased, due to the increased service load.

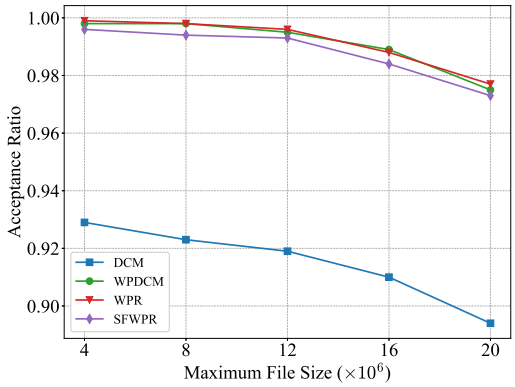
Figure 5 presents the performance of different schemes when the maximum computation load shifted from 1×10^8 CPU cycles to 4×10^8 CPU cycles. Similar to the observations obtained from Figure 4, it was observed that all schemes achieved higher acceptance ratios when the computation load of tasks varied in a small range. Due to the small range of computation loads, SFWPR outperforms DCM in the uniform distribution (Figure 5a) and the Zipf distribution prioritizing small input files (Figure 5b). However, as the maximum computation load increases, the performance of SFWPR degrades rapidly, and it becomes inferior to DCM in the Zipf distribution prioritizing large input files case (Figure 5c). This is because the minor difference in computation load meant the refilling policy $\mathbb{R}$ for SFWPR and WPR were approximately the same. Moreover, the evident degradation shown in Figure 5c suggests that SFWPR was sensitive to computing load.

Figure 6 depicts the achieved acceptance ratio of various schemes while varying the system bandwidth from 6×10^6 hertz to 1.8×10^7 hertz. It was observed that a larger bandwidth led to a higher acceptance ratio for all schemes. The primary reason for this phenomenon is that the over-provisioned bandwidth reduced the pressure on computing resources. With shorter transmission delay, there is a more considerable margin of compensation for computational delay. As a result, SFWPR outperforms DCM due to the additional supplement of system bandwidth, as compared to the results presented in Figure 2.

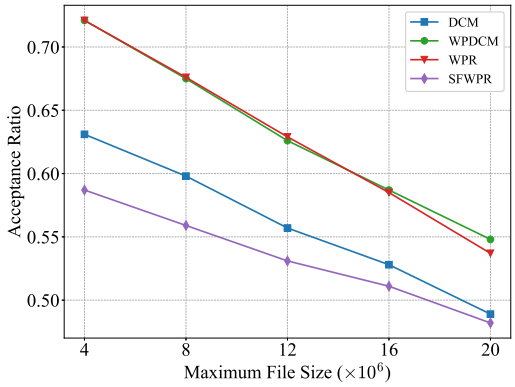
Figure 7 illustrates how the performance of the schemes mentioned above differed as the edge computational capacity increased from 5×10^9 CPU cycles per second to 30×10^9 CPU cycles per second. It can be concluded that the performance of SFWPR degraded rapidly as the computational capacity became smaller. This is attributed to its refilling policy $\mathbb{R}$, which inherently required more computing resources than the WPR scheme. Moreover, the performance gap between the WPR-based schemes (WPDCM, WPR) and DCM widened as the edge computational capacity increased, indicating the superior flexibility of WPR-based schemes in resource allocation.



(a)

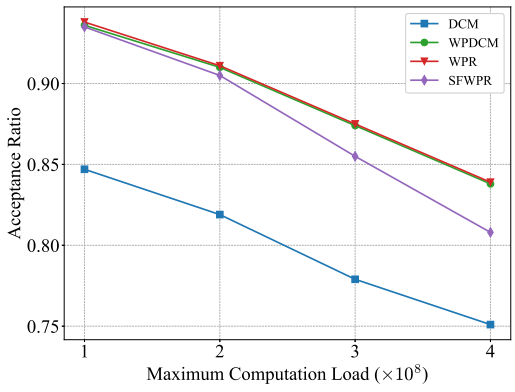


(b)

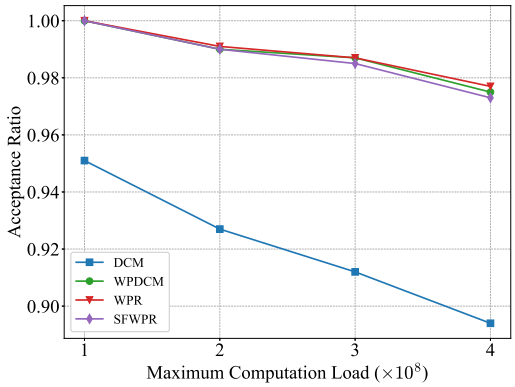


(c)

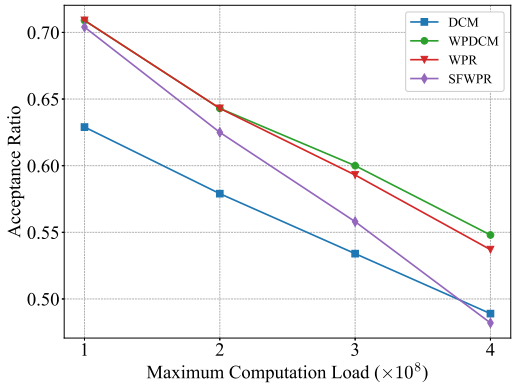
Figure 4. Acceptance ratio versus maximum file size: (**a**) uniform distribution; (**b**) Zipf distribution prioritizing small input files; (**c**) Zipf distribution prioritizing large input files.



(a)



(b)



(c)

Figure 5. Acceptance ratio versus maximum computation load: (**a**) uniform distribution; (**b**) Zipf distribution prioritizing small input files; (**c**) Zipf distribution prioritizing large input files.

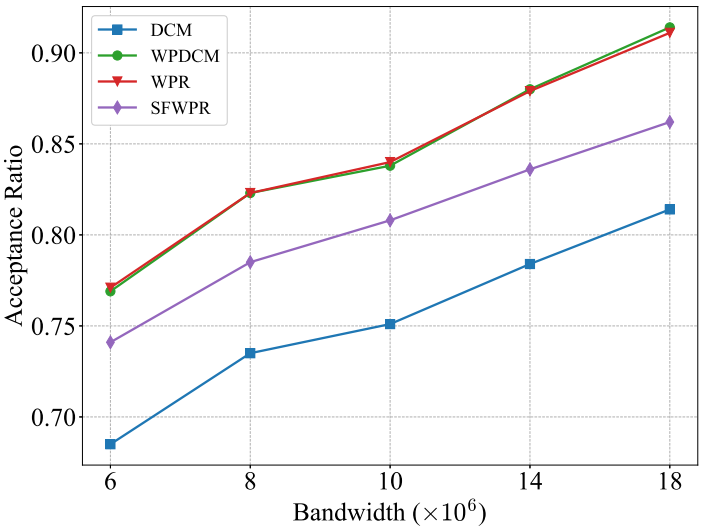


Figure 6. Acceptance ratio versus system bandwidth.

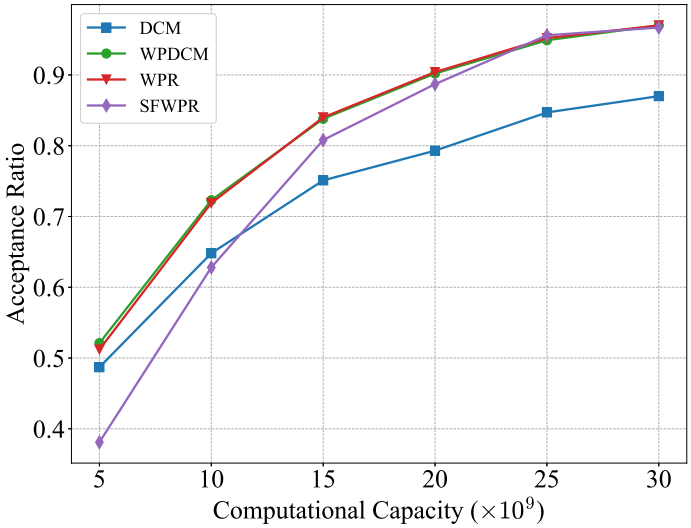


Figure 7. Acceptance ratio versus edge computational capacity.

6. Conclusions

The current minimum latency scheme is inadequate in fully utilizing resources of the asymmetrical system to provide satisfactory computation offloading service for UDs with a maximum request acceptance ratio. In this paper, we propose a water pumping and refilling algorithm that exploits the margin between the delay-minimum scheme and the near-deadline scheme to achieve the maximal acceptance ratio. We first solve the resource allocation sub-problem of the delay-minimum scheme, which provided inspiration for the design of the water pumping and refilling algorithm. The water pumping algorithm can function not only as a supplementary algorithm to a given scheme, but also as a standalone algorithm to obtain the resource allocation scheme following a customizable refilling policy $\mathbb{R}$. The simulation results demonstrated that our proposed algorithm outperforms delay-minimum schemes in achieving a high acceptance ratio.

In the future, we plan to investigate computation offloading and resource allocation schemes under energy consumption constraints. Additionally, we will explore system models with collaborations between multiple base stations.

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

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Appendix A

Proof. The resulting Karush–Kuhn–Tucker (KKT) conditions can be denoted as follows:

$$\frac{\partial L}{\partial \alpha_m} = \lambda_1 - \frac{(1+\nu_m) l_m}{\alpha_m^2 \overline{R}_m} = 0, \forall m \in \mathcal{M}_1, \tag{A1}$$

$$\frac{\partial L}{\partial \beta_m} = \lambda_2 - \frac{(1+\nu_m) L_m}{\beta_m^2 Fe} = 0, \forall m \in \mathcal{M}_1, \tag{A2}$$

$$\sum_{m \in \mathcal{M}_1} \alpha_m - 1 \leq 0, \tag{A3}$$

$$\lambda_1 (\sum_{m \in \mathcal{M}_1} \alpha_m - 1) = 0, \tag{A4}$$

$$\sum_{m \in \mathcal{M}_1} \beta_m - 1 \leq 0, \tag{A5}$$

$$\lambda_2 (\sum_{m \in \mathcal{M}_1} \beta_m - 1) = 0, \tag{A6}$$

$$\nu_m (\frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe} - T_m^{dl}) = 0, \forall m \in \mathcal{M}_1, \tag{A7}$$

$$\frac{l_m}{\alpha_m \overline{R}_m} + \frac{L_m}{\beta_m Fe} - T_m^{dl} \leq 0, \tag{A8}$$

$$\lambda_1, \lambda_2 \geq 0. \tag{A9}$$

With some manipulation to Equations (A1) and (A2), we can obtain the bandwidth and computing resources allocation solution as follows:

$$\alpha_m = \frac{\sqrt{\frac{(1+\nu_m) l_m}{\overline{R}_m}}}{\sqrt{\lambda_1}}, \tag{A10}$$

$$\beta_m = \frac{\sqrt{\frac{(1+\nu_m) L_m}{Fe}}}{\sqrt{\lambda_2}}. \tag{A11}$$

With the contradiction technique adopted in Appendix B in [7], we have:

$$\sum_{m \in M_1} \alpha_m - 1 = 0, \tag{A12}$$

$$\sum_{m \in M_1} \beta_m - 1 = 0. \tag{A13}$$

Substituting Equation (A10) and Equation (A11) into Equation (A12) and Equation (A13) respectively, the Lagrange multipliers $\sqrt{\lambda_1}, \sqrt{\lambda_2}$ can be derived as:

$$\sqrt{\lambda_1} = \sum_{i \in M_1} \sqrt{\frac{(1+\nu_i) l_i}{\overline{R}_i}}, \tag{A14}$$

$$\sqrt{\lambda_2} = \sum_{j \in M_1} \sqrt{\frac{(1+\nu_j) L_j}{Fe}}. \tag{A15}$$

Finally, combining Equation (A10) with Equation (A14) and Equation (A11) with Equation (A15), the bandwidth and computing resources allocation scheme to Problem (P2.1) can be denoted as follows:

$$(\alpha_m, \beta_m) = \left(\frac{\sqrt{\frac{(1+\nu_m) l_m}{\overline{R}_m}}}{\sum\limits_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) l_i}{\overline{R}_i}}}, \frac{\sqrt{\frac{(1+\nu_m) L_m}{Fe}}}{\sum\limits_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) L_i}{Fe}}} \right), \forall m \in \mathcal{M}_1. \tag{A16}$$

This ends the proof. □

Appendix B

Proof. The sum of upload transmission time of UDs in $\mathcal{M}_1$ can be written as follows

$$\sum_{m \in M_1} \frac{l_m}{\alpha_m \overline{R}_m} = \sum_{m \in M_1} \frac{l_m}{\alpha_m \overline{R}_m} \sum_{m \in M_1} \alpha_m \geq \left(\sum_{m \in M_1} \sqrt{\frac{l_m}{\alpha_m \overline{R}_m}} \sqrt{\alpha_m} \right)^2 = \left(\sum_{m \in M_1} \sqrt{\frac{l_m}{\overline{R}_m}} \right)^2. \tag{A17}$$

The first equality comes from Equation (A12) and the inequality comes from Cauchy–Buniakowsky–Schwarz inequality where the equality holds for all UDs in $\mathcal{M}_1$:

$$\frac{\sqrt{\frac{l_1}{\alpha_1 \overline{R}_1}}}{\sqrt{\alpha_1}} = \frac{\sqrt{\frac{l_2}{\alpha_2 \overline{R}_2}}}{\sqrt{\alpha_2}} = \cdots = \frac{\sqrt{\frac{l_m}{\alpha_m \overline{R}_m}}}{\sqrt{\alpha_m}} = c, \tag{A18}$$

where c is a constant. With some manipulation, we have

$$\sqrt{\frac{l_m}{{\alpha_m}^2 \overline{R}_m}} = \frac{\sqrt{\lambda_1}}{\sqrt{(1+\nu_m)}} = \frac{\sum\limits_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) l_i}{\overline{R}_i}}}{\sqrt{(1+\nu_m)}}, \tag{A19}$$

where the first equality comes from Equation (A10) and the second equality comes from Equation (A16). Thus, to achieve the minimum of Equation (A17), we need to determine the values of $\nu_m, \forall m \in \mathcal{M}_1$ to guarantee $\frac{\sqrt{\lambda_1}}{\sqrt{(1+\nu_m)}} = c$. An intuitive scheme is to assign $\nu_m, \forall m \in \mathcal{M}_1$ with equal values (for example, 1).

Similarly, we can conclude that the sum of edge processing time $\sum\limits_{m \in M_1} \frac{L_m}{\beta_m Fe}$ achieves its minimum $\left(\sum\limits_{m \in M_1} \sqrt{\frac{L_m}{Fe}} \right)^2$ if:

$$\sqrt{\frac{L_m}{{\beta_m}^2 Fe}} = \frac{\sqrt{\lambda_2}}{\sqrt{(1+\nu_m)}} = \frac{\sum\limits_{i \in \mathcal{M}_1} \sqrt{\frac{(1+\nu_i) L_i}{Fe}}}{\sqrt{(1+\nu_m)}} = d, \forall m \in \mathcal{M}_1, \tag{A20}$$

where d is a constant. Thus, setting equal value to $\nu_m \quad \forall m \in \mathcal{M}_1$ can obtain the minimum delay cost $\left(\sum_{m\in M_1}\sqrt{\frac{l_m}{R_m}}\right)^2+\left(\sum_{m\in M_1}\sqrt{\frac{L_m}{Fe}}\right)^2, \forall m \in \mathcal{M}_1$. Accordingly, the optimal Lagrange multipliers λ_1, λ_2 and resource allocation decisions can be denoted as

$$\lambda_1^* = \left(\sum_{i\in\mathcal{M}_1}\sqrt{\frac{l_i}{R_i}}\right)^2, \qquad \lambda_2^* = \left(\sum_{i\in\mathcal{M}_1}\sqrt{\frac{L_i}{Fe}}\right)^2, \tag{A21}$$

$$\alpha_m^* = \frac{\sqrt{\frac{l_m}{R_m}}}{\sum_{i\in\mathcal{M}_1}\sqrt{\frac{l_i}{R_i}}}, \qquad \beta_m^* = \frac{\sqrt{\frac{L_m}{Fe}}}{\sum_{i\in\mathcal{M}_1}\sqrt{\frac{L_i}{Fe}}}. \tag{A22}$$

This ends the proof. □

References

1. Dai, Y.; Xu, D.; Maharjan, S.; Zhang, Y. Joint computation offloading and user association in multi-task mobile edge computing. *IEEE Trans. Veh. Technol.* **2018**, *67*, 12313–12325. [CrossRef]
2. Hu, Y.C.; Patel, M.; Sabella, D.; Sprecher, N.; Young, V. Mobile edge computing—A key technology towards 5G. *ETSI White Pap.* **2015**, *11*, 1–16.
3. Lin, L.; Liao, X.; Jin, H.; Li, P. Computation Offloading Toward Edge Computing. *Proc. IEEE* **2019**, *107*, 1584–1607. [CrossRef]
4. Khan, A.A.; Laghari, A.A.; Shaikh, Z.A.; Dacko-Pikiewicz, Z.; Kot, S. Internet of Things (IoT) security with blockchain technology: A state-of-the-art review. *IEEE Access* **2022**, *10*, 122679–122695. [CrossRef]
5. Wang, Y.; Sheng, M.; Wang, X.; Wang, L.; Li, J. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Trans. Commun.* **2016**, *64*, 4268–4282. [CrossRef]
6. Zhang, J.; Hu, X.; Ning, Z.; Ngai, E.C.H.; Zhou, L.; Wei, J.; Cheng, J.; Hu, B. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. *IEEE Internet Things J.* **2017**, *5*, 2633–2645. [CrossRef]
7. Ren, J.; Yu, G.; He, Y.; Li, G.Y. Collaborative cloud and edge computing for latency minimization. *IEEE Trans. Veh. Technol.* **2019**, *68*, 5031–5044. [CrossRef]
8. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
9. Shakarami, A.; Ghobaei-Arani, M.; Masdari, M.; Hosseinzadeh, M. A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective. *J. Grid Comput.* **2020**, *18*, 639–671. [CrossRef]
10. Yuan, H.; Bi, J.; Zhou, M. Geography-aware task scheduling for profit maximization in distributed green data centers. *IEEE Trans. Cloud Comput.* **2020**, *10*, 1864–1874. [CrossRef]
11. Yuan, H.; Zhou, M. Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems. *IEEE Trans. Autom. Sci. Eng.* **2020**, *18*, 1277–1287. [CrossRef]
12. Wang, Y.; Tao, X.; Zhang, X.; Zhang, P.; Hou, Y.T. Cooperative task offloading in three-tier mobile computing networks: An ADMM framework. *IEEE Trans. Veh. Technol.* **2019**, *68*, 2763–2776. [CrossRef]
13. Wei, Y.; Yu, F.R.; Song, M.; Han, Z. Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning. *IEEE Internet Things J.* **2018**, *6*, 2061–2073. [CrossRef]
14. Khan, A.A.; Laghari, A.A.; Shafiq, M.; Awan, S.A.; Gu, Z. Vehicle to Everything (V2X) and Edge Computing: A Secure Lifecycle for UAV-Assisted Vehicle Network and Offloading with Blockchain. *Drones* **2022**, *6*, 377. [CrossRef]
15. Mohajer, A.; Daliri, M.S.; Mirzaei, A.; Ziaeddini, A.; Nabipour, M.; Bavaghar, M. Heterogeneous computational resource allocation for NOMA: Toward green mobile edge-computing systems. *IEEE Trans. Serv. Comput.* **2022**, *16*, 1225–1238. [CrossRef]
16. Zhang, W.; Wen, Y.; Guan, K.; Kilper, D. Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel. *IEEE Trans. Wirel. Commun.* **2013**, *12*, 4569–4581. [CrossRef]
17. Chen, J.; Xing, H.; Lin, X.; Nallanathan, A.; Bi, S. Joint Resource Allocation and Cache Placement for Location-Aware Multi-User Mobile-Edge Computing. *IEEE Internet Things J.* **2022**, *9*, 25698–25714. [CrossRef]
18. Dai, Y.; Zhang, K.; Maharjan, S.; Zhang, Y. Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12175–12186. [CrossRef]
19. Huang, P.Q.; Wang, Y.; Wang, K.; Liu, Z.Z. A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing. *IEEE Trans. Cybern.* **2019**, *50*, 4228–4241. [CrossRef]
20. Zeng, S.; Huang, X.; Li, D. Joint Communication and Computation Cooperation in Wireless Powered Mobile Edge Computing Networks with NOMA. *IEEE Internet Things J.* **2023**. [CrossRef]
21. Karimi, E.; Chen, Y.; Akbari, B. Task offloading in vehicular edge computing networks via deep reinforcement learning. *Comput. Commun.* **2022**, *189*, 193–204. [CrossRef]

22. Zhou, F.; Hu, R.Q. Computation efficiency maximization in wireless-powered mobile edge computing networks. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 3170–3184. [CrossRef]
23. Mukherjee, M.; Kumar, V.; Zhang, Q.; Mavromoustakis, C.X.; Matam, R. Optimal Pricing for Offloaded Hard-and Soft-Deadline Tasks in Edge Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 9829–9839. [CrossRef]
24. Yan, J.; Bi, S.; Duan, L.; Zhang, Y.J.A. Pricing-driven service caching and task offloading in mobile edge computing. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 4495–4512. [CrossRef]
25. Wang, Q.; Guo, S.; Liu, J.; Pan, C.; Yang, L. Profit maximization incentive mechanism for resource providers in mobile edge computing. *IEEE Trans. Serv. Comput.* **2019**, *15*, 138–149. [CrossRef]
26. Zhou, J.; Zhang, X. Fairness-aware task offloading and resource allocation in cooperative mobile-edge computing. *IEEE Internet Things J.* **2021**, *9*, 3812–3824. [CrossRef]
27. Hejja, K.; Berri, S.; Labiod, H. Network slicing with load-balancing for task offloading in vehicular edge computing. *Veh. Commun.* **2022**, *34*, 100419. [CrossRef]
28. Meng, J.; Tan, H.; Li, X.Y.; Han, Z.; Li, B. Online deadline-aware task dispatching and scheduling in edge computing. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *31*, 1270–1286. [CrossRef]
29. Ren, J.; Yu, G.; Cai, Y.; He, Y. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 5506–5519. [CrossRef]
30. Bi, S.; Huang, L.; Wang, H.; Zhang, Y.J.A. Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7519–7537. [CrossRef]
31. Qiu, X.; Zhang, W.; Chen, W.; Zheng, Z. Distributed and collective deep reinforcement learning for computation offloading: A practical perspective. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1085–1101. [CrossRef]
32. Zhou, H.; Jiang, K.; Liu, X.; Li, X.; Leung, V.C. Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing. *IEEE Internet Things J.* **2021**, *9*, 1517–1530. [CrossRef]
33. Zhu, X.; Luo, Y.; Liu, A.; Bhuiyan, M.Z.A.; Zhang, S. Multiagent deep reinforcement learning for vehicular computation offloading in IoT. *IEEE Internet Things J.* **2020**, *8*, 9763–9773. [CrossRef]
34. Ndikumana, A.; Tran, N.H.; Ho, T.M.; Han, Z.; Saad, W.; Niyato, D.; Hong, C.S. Joint communication, computation, caching, and control in big data multi-access edge computing. *IEEE Trans. Mob. Comput.* **2019**, *19*, 1359–1374. [CrossRef]
35. Wen, W.; Cui, Y.; Quek, T.Q.; Zheng, F.C.; Jin, S. Joint optimal software caching, computation offloading and communications resource allocation for mobile edge computing. *IEEE Trans. Veh. Technol.* **2020**, *69*, 7879–7894. [CrossRef]
36. Gong, Y.; Yao, H.; Wang, J.; Li, M.; Guo, S. Edge intelligence-driven joint offloading and resource allocation for future 6G industrial internet of things. *IEEE Trans. Netw. Sci. Eng.* **2022**. [CrossRef]
37. Liu, Y.; Tan, Z.; Hu, H.; Cimini, L.J.; Li, G.Y. Channel estimation for OFDM. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1891–1908. [CrossRef]
38. Dong, L.; He, W.; Yao, H. Task Offloading and Resource Allocation for Tasks with Varied Requirements in Mobile Edge Computing Networks. *Electronics* **2023**, *12*, 366. [CrossRef]
39. Bi, S.; Zhang, Y.J. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Trans. Wirel. Commun.* **2018**, *17*, 4177–4190. [CrossRef]
40. Xu, X.; Li, Y.; Huang, T.; Xue, Y.; Peng, K.; Qi, L.; Dou, W. An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks. *J. Netw. Comput. Appl.* **2019**, *133*, 75–85. [CrossRef]
41. Fang, C.; Xu, H.; Yang, Y.; Hu, Z.; Tu, S.; Ota, K.; Yang, Z.; Dong, M.; Han, Z.; Yu, F.R.; et al. Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks. *IEEE Internet Things J.* **2022**, *9*, 16874–16883. [CrossRef]
42. Fang, C.; Liu, C.; Wang, Z.; Sun, Y.; Ni, W.; Li, P.; Guo, S. Cache-assisted content delivery in wireless networks: A new game theoretic model. *IEEE Syst. J.* **2020**, *15*, 2653–2664. [CrossRef]