future internet

MDPI

Article

Computation Offloading Based on a Distributed Overlay Network Cache-Sharing Mechanism in Multi-Access Edge Computing

Yazhi Liu [1,†], Pengfei Zhong [1,†], Zhigang Yang [2,*], Wei Li [1] and Siwei Li [1]

[1] College of Artificial Intelligence, North China University of Science and Technology, Tangshan 063210, China; liuyazhi@ncst.edu.cn (Y.L.); zhongpengfei@stu.ncst.edu.cn (P.Z.)
[2] College of Electrical Engineering, North China University of Science and Technology, Tangshan 063210, China
* Correspondence: yzg@ncst.edu.cn
† These authors contributed equally to this work.

Abstract: Multi-access edge computing (MEC) enhances service quality for users and reduces computational overhead by migrating workloads and application data to the network edge. However, current solutions for task offloading and cache replacement in edge scenarios are constrained by factors such as communication bandwidth, wireless network coverage, and limited storage capacity of edge devices, making it challenging to achieve high cache reuse and lower system energy consumption. To address these issues, a framework leveraging cooperative edge servers deployed in wireless access networks across different geographical regions is designed. Specifically, we propose the Distributed Edge Service Caching and Offloading (DESCO) network architecture and design a decentralized resource-sharing algorithm based on consistent hashing, named Cache Chord. Subsequently, based on DESCO and aiming to minimize overall user energy consumption while maintaining user latency constraints, we introduce the real-time computation offloading (RCO) problem and transform RCO into a multi-player static game, prove the existence of Nash equilibrium solutions, and solve it using a multi-dimensional particle swarm optimization algorithm. Finally, simulation results demonstrate that the proposed solution reduces the average energy consumption by over 27% in the DESCO network compared to existing algorithms.

Keywords: multi-access edge computing; computation offloading; consistent hashing; P2P overlay network; multi-dimensional discrete PSO

Citation: Liu, Y.; Zhong, P.; Yang, Z.; Li, W.; Li, S. Computation Offloading Based on a Distributed Overlay Network Cache-Sharing Mechanism in Multi-Access Edge Computing. *Future Internet* **2024**, *16*, 136. https://doi.org/10.3390/fi16040136

Academic Editor: Paolo Bellavista

Received: 19 March 2024
Revised: 8 April 2024
Accepted: 17 April 2024
Published: 19 April 2024

1. Introduction

In recent years, with the rapid proliferation of wireless IoT applications such as connected vehicles, VR/AR, smart cities, and personalized streaming videos, there has been a surge in computation-sensitive and real-time services in mobile networks [1]. These services typically require completion within a short time frame (20–125 ms) [2]. Figure 1 illustrates the workflow of real-time services. Generally, three steps are involved: generation, processing, and transmission [3]. Firstly, mobile users (MUs) such as connected vehicles, AR/VR devices, mobile phones, or drones would continuously generate sensor or user data (infrared, radar, video streams, health, and so on) when requesting specific real-time services. Subsequently, MUs can process these tasks locally or transmit them to server units located in the cloud or edge to obtain computing services. The computing operations will be carried out with the assistance of application data (executable programs). The computing results return to the MUs at the end of the time frame. For example, vehicles requiring autonomous driving services would detect environment data from fusion monitoring devices (in-vehicle cameras, millimeter-wave or ultrasonic radar, etc.) while driving. After preprocessing, these raw data are sent to computing units deployed with corresponding target detection algorithms (or processed directly by locally embedded computing units) to perceive road conditions and provide optimal driving decisions.

This necessitates an efficient, cost-effective service model that minimizes user energy consumption while ensuring service quality. Traditional cloud computing architectures, as a centralized solution, encounter challenges such as severe wide-area network latency and fluctuating service quality when handling these new types of services.

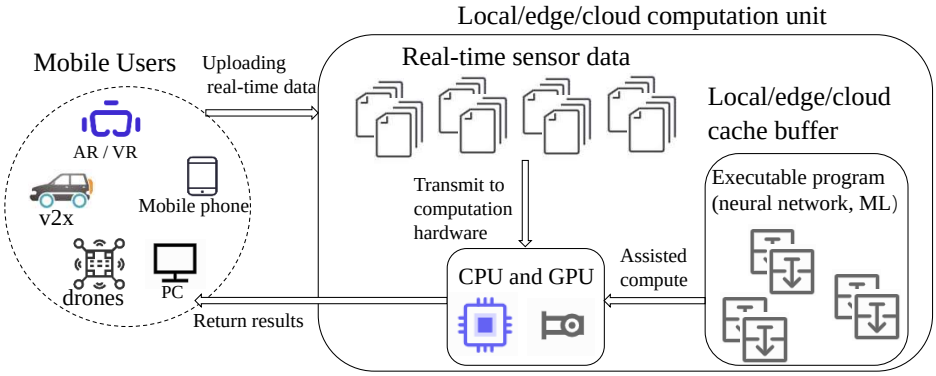

Figure 1. The computation and processing workflow of online real-time services.

Multi-access edge computing (MEC) [4] emerges as a computing paradigm capable of addressing these issues. It leverages the distributed computing power and communication resources at the network edge, particularly at the edge of the mobile wireless access network (RAN), by deploying communication entities such as road side units (RSUs), smart gateways, and small cell base stations to construct a network topology. It deploys corresponding computing and storage units at wireless access points (WAPs) to provide personalized services to MUs. MUs can directly access computing services from edge servers located at WAPs, avoiding traffic bottlenecks in the core and backhaul network during routing, thus partially alleviating the cache and computational burdens on data centers.

Additionally, MEC offers performance-optimizing technologies such as computation offloading (CO) [5] and edge caching (EC) [6]. Computation offloading, as a service optimization paradigm, allows MUs to delegate complex interactive computing tasks to edge nodes and receive computation results via the wireless downlinks between WAPs and MUs, thereby greatly alleviating the problem of high computational energy consumption in terminal devices with limited computational resources during service usage. Edge caching allows users to cache application data at communication nodes, enabling different MUs requesting the same service to directly upload locally generated real-time data for computation, avoiding the redundant transmission of the same application data and thus alleviating high latency and heavy load in the fronthaul network.

As an evolution of traditional mobile base stations, the MEC architecture allows for the scheduling and management of parallel communication resources at the network edge layer. This provides a new solution for designing cache data-sharing strategies among MEC nodes using backhaul networks, further optimizing system performance. It also offers cloud service providers more flexible computation offloading solutions by changing the execution and storage locations of data to optimize system performance.

Although the MEC network paradigm can overcome some of the drawbacks of cloud computing and meet the service constraints of real-time applications, there are still many challenges. Specifically, from the perspective of data sharing, in the current edge computing domain, the most well-known approach is based on blockchain [7] for distributed data storage and synchronized data sharing. However, this approach faces serious performance issues for high-throughput applications due to the low performance of blockchain networks, high costs associated with storing block data, and scalability challenges posed by large-scale data synchronization. These issues present difficulties in scheduling real-time applications.

To address these issues, our work focuses on leveraging performance optimization methods in MEC networks to provide real-time services for MUs. The main contributions of this study can be summarized as follows:

- The thesis proposes a Distributed Edge Service Caching and Offloading (DESCO) architecture based on edge servers' collaboration to provide real-time services for mobile users. Within DESCO, the optimization problem of minimizing the long-term average energy consumption of users while satisfying latency constraints is established, for which cache sharing, replacement, and computation offloading methods are leveraged to further optimize user costs.
- A decentralized consistent hashing-based cache-sharing mechanism named "Cache Chord" is designed. This mechanism leverages the communication backhaul links at the edge layer and expands the DESCO framework by allowing edge servers to self-organize into a circular logical topology, facilitating application data sharing. Additionally, a cache IP list mechanism is devised to link application resource key-values in the overlay network with actual data in the real network.
- The real-time computation offloading (RCO) problem is transformed into a multi-player static game among MUs within the wireless network coverage of each server in order to reduce the current time slot energy consumption. A multi-dimensional discrete particle swarm optimization algorithm is applied to solve this problem after proving the existence of the Nash equilibrium (NE) solution for the game. Furthermore, the exploration coefficients and iteration rules of the algorithm are designed to meet environment constraints.
- Finally, simulation experiments are conducted to evaluate the performance of the proposed framework and algorithms. Results demonstrate that the proposed offloading method effectively reduces overall energy consumption at the user layer and obtains better converges compared to baseline algorithms.

The remainder of the paper is organized as follows: Section 2 introduces the related work of computation offloading and cache-sharing technology. Section 3 describes the system model of DESCO, and further introduces the cache-sharing mechanism. In Section 4, we transfer the RCO problem into a static game and proof the existence of NE solutions, then design the MDPSO algorithm. Section 5 presents simulation experiments to explore the superiority of the MDPSO algorithm and Cache Chord. Finally, the conclusion is given in Section 6.

2. Related Works

With the development of real-time applications, resource requests generated by terminal devices tend to be independent and the service requests from MUs exhibit more dynamic characteristics. Specifically, as noted in [8,9], within MU groups, the popularity of services follows a Zipf distribution, implying that a single server-based service requires maintaining a large cache database to improve hit rates, leading to significantly increase system redundancy. Simultaneously, computation offloading services impose considerable pressure on the transmission bandwidth of the wireless uplink from end to edge, while the computational capabilities of terminal MU devices are limited and consume a considerable amount of energy [10]. Therefore, coordinating energy consumption and service efficiency to maximize the system's offloading benefits is a worthy issue to explore. Given the current state of limited infrastructure capacity and the limited computational capabilities of user devices in edge networks, the effective use of network resources to improve resource utilization and system performance has become an urgent challenge for network service providers.

To address this situation, researchers primarily employ two methods: firstly, by designing fine-grained and efficient computation offloading algorithms, and secondly, by considering the heterogeneity of edge devices and their geographical distribution characteristics, leveraging collaborative services between edges to enable on-demand resource mobility and fully utilize existing computing devices to ensure service quality. The following two subsections will provide detailed explanations of these approaches.

2.1. Computation Offloading Strategy

The real-time computation offloading (RCO) can enhance the QoE of interactive gaming [11]. In this scenario, MEC servers would process real-time action data offloaded by players (MUs) and render them into corresponding in-game scenes. The rendered data are then compressed via a video encoder and transmitted back to users through video-streaming. The RCO strategy allows servers to wisely offload service requests from multiple users, maximizing overall user satisfaction. RCO can also be applied in energy-constrained drones [12] and wearable devices [13] to reduce the computational overhead of local devices. Complex neural networks or machine learning computations can drain battery life for these devices. Additionally, requests from similar types of devices often exhibit popularity (e.g., health monitoring, object detection, and path planning algorithms). The RCO algorithm can select a strategy that maximizes offloading benefits based on the application categories stored on the server and the user cluster's requests, thereby reducing the computational energy consumption of MUs and extending device battery life.

Multiple mobile users can achieve resource-sharing and collaborative computing, while nearby MUs may request similar tasks [14]. Based on this scenario, a fine-grained collaborative computing offloading and caching strategy is proposed to minimize the overall execution latency of MUs within the network [15]. Additionally, the concept of a call graph is utilized to model the offloading and caching relationships among MUs. It is noteworthy that [16] considers software-fetching and multicasting in network modeling, mathematically characterizing the processes of data uploading, task execution, and computation result downloading to minimize cache and weighted deadline as optimization objectives. They employ a joint algorithm combining ADMM and a penalty convex–concave procedure to obtain the optimal offloading strategy.

The ADMM algorithm is also applied in [17] to obtain distributed offloading decisions. In this work, the authors propose a computation offloading scheme where computational tasks generated by ground users can be computed locally, on Low Earth Orbit (LEO) satellites, or on cloud servers. The authors also consider the limited computational capabilities and coverage time of each LEO satellite. They subsequently investigate the optimization problem of minimizing the total energy consumption of ground users, which is discrete and non-convex, and convert it into a linear programming problem.

In Time-Division Multiple Access (TDMA)-based MEC systems, a partial offloading strategy based on an iterative heuristic algorithm is proposed [18] to minimize the total energy consumption of MUs while ensuring the task delay constraints. This strategy jointly optimizes task offloading rates, channel allocation, and MEC computing resource allocation. The authors decompose the problem into a series of offloading subproblems and design a two-stage algorithm to iteratively solve the offloading task set until achieving the minimum energy consumption.

When addressing performance optimization problems with computation offloading algorithms, it is observed that the discussion on the economies of scale brought by multi-server clusters is insufficient. Most articles only conduct research based on single-server scenarios, while in reality, MEC servers are often densely deployed in scenarios such as streets, malls, and schools to provide services to users. This cluster effect can be applied to the design of distributed network models and data-sharing algorithms. Wired backhaul links between edges can transmit large amounts of real-time data, which are also beneficial for the on-demand allocation of cached content related to real-time tasks.

2.2. Data-Sharing Mechanism

The authors of [19,20] focus on algorithm design to explore the cache-sharing strategy in distributed edge environments. To address the problem of MUs' difficulty in discovering required IoT resources due to device heterogeneity, a Fog Computing-based resource discovery solution named FDS-RD is proposed [19]. FDS-RD employs a Distributed Hash Table (DHT) to map resources to a structured peer-to-peer (P2P) network, facilitating resource discovery in large-scale IoT environments.

VCNS [20] is a content-sharing network, tailored for the vehicular ad hoc network (VANET) scenario. It presents an edge caching scheme based on cross-entropy and dynamically adjusts caching based on content popularity within the request scope. Additionally, it designs a collaborative content delivery mechanism for RSUs to further reduce system latency overhead. Meanwhile, the authors of [21,22] focus on network logical topology design, exploring both structured and unstructured network topologies. The emphasis is on managing resource nodes to organize decentralized sharing networks, providing cached content to users to optimize system overheads.

Considering cooperative caching among edge servers, ref. [21] proposes a distributed edge data indexing system called EDIndex. In this system, any server maintains a hierarchical counting bloom filter (HCBF) tree index structure, which indexes the data stored in nearby edge servers, enabling fast querying at the edge. SA-Chord [22] is a pure edge computing-based adaptive distributed overlay network architecture. Based on the chord protocol, it designs a two-layer circular routing overlay composed of peer nodes and super nodes. Peer nodes only participate in content reception and transmission without routing, while super node clusters are responsible for maintaining routing tables for message-forwarding, achieving decentralized content-sharing based on the dual-layer structure.

From the above works, it can be observed that decentralized application caching and retrieval based on distributed hash can efficiently achieve data sharing, effectively reducing various overheads of the system, considering the limited storage capacity of geographically distributed edge service nodes. Therefore, this paper will explore the optimal task scheduling strategy based on a heuristic algorithm. Additionally, it will leverage the edge backhaul links' path in existing multi-server–multi-user MEC networks to construct a structured data-sharing mechanism based on DHT.

3. System Descriptions and Assumptions

3.1. DESCO Network Model

We designed an Edge Service Caching and Offloading (DESCO) architecture based on the multi-server–multi-user end–edge two-layer network paradigm [22], as illustrated in Figure 2. In the DESCO network, the end layer consists of MU clusters, distributed across the coverage areas of various wireless networks, organized by different edge servers, and each of them is connected to the nearest server to obtain computing and offloading services. In the edge layer, MEC servers are deployed within wireless access points (WAPs) and equipped with computing and storage units. MEC servers communicate with each other through wired backhaul links and are self-organized at the application layer into a decentralized ring network called Cache Chord. The maintenance of this network structure relies solely on the consistent hashing protocol followed by server nodes, with detailed design discussed in Section 3.3.

Three computing modes exist in the DESCO network: local computing, pure offloading, and cache-based offloading. Local computing means MUs processing real-time tasks by themselves; pure offloading signifies that the MEC server's local cache pool does not store the application programs required to execute user requests, so MUs need to offload real-time data and application data to the server. Cache-based offloading indicates that the server stores the application data for the task, and all users requesting the service only need to offload real-time data to the server, reducing some of the transmission energy consumption. This will be elaborated further in Section 3.4.

Table 1 summarizes the main parameters included in this study. The total number of MEC servers joining the DESCO network edge layer is denoted by the set $\Delta = \{1, 2, 3, ..., D\}$. Each distributed MEC server node uses its IP address as a unique identifier, represented by the set NID $= \{nip_1, nip_2, ..., nip_D\}$. The set KID $= \{kid_1, kid_2, ..., kid_K\}$ signifies unique resource identifiers for applications, utilized for identifying and indexing services within the Cache Chord network. Here, the subscript K indicates the total number of tasks that any user n at different geographical locations might request at any moment.

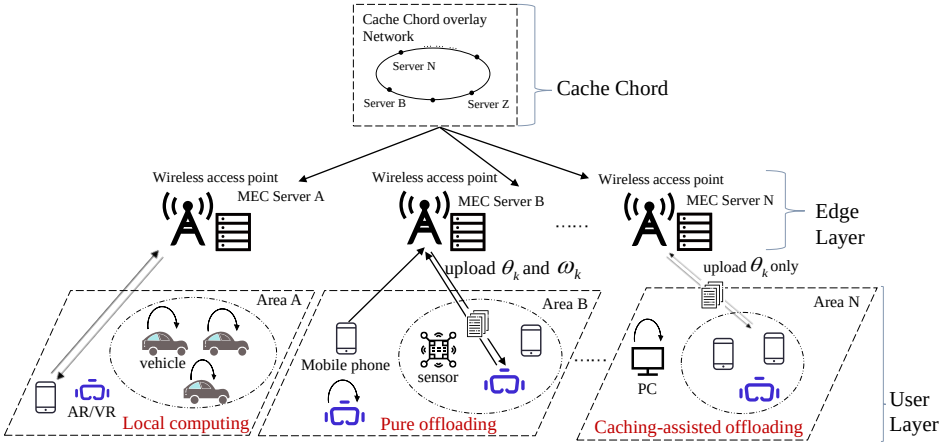

Figure 2. DESCO network architecture.

Table 1. Notation summary.

Notation	Definition
T,τ	Set of time sequence; time frame length
θ_k, ω_k, ϕ_k	Real-time data volume of k; application data size of k; necessary FLOPs of task k
Δ, D	MEC server set; number of MEC server
Nd, N	User request under server d; request of user n
$o_{d,n}$	MUs' local compute power
K, NID, KID	Service feature set; MEC server IP address set; application identifier set
$U^t_{\mathrm{N}_d}$, μ^t_n	User request under server d; request of user n
b	Wireless subchannel bandwidth of server d
$l^{d,t}_n$	Minimum efficient uplink transmission rate of user k in range d
ψ_n	Transmit power of user n
$\mathrm{V}^{d,t}$	RCO decision of server d in slot t
$v^{d,t}_n$	RCO policy of user n
$\mathrm{P}^{d,t}$	Application cache replacement operation of d
$\rho^{d,t}_k$	Operation of task k in server d at time t
$\mathrm{Y}_{n,d,k,t}$	Energy consumption of user n in range d

In the DESCO system, every entity operates in the time series $\mathrm{T} = \{1, 2, ..., T\}$. This value serves as the fundamental unit for both the system's offloading and caching decisions and the variations in user task requests. The duration of any arbitrary time slot $t \in \mathrm{T}$ is set as a constant value, denoted by τ, representing the latency constraint of real-time applications. This means that at time t, when an MU generates a service request and produces corresponding real-time data, the computation result must be returned before the end of time t.

Let $\mathrm{K} = \{\langle\theta_1, \omega_1, \phi_1\rangle, \langle\theta_2, \omega_2, \phi_2\rangle, ..., \langle\theta_K, \omega_K, \phi_K\rangle\}$ be the set of task features, where a task $k \in \{1, 2, .., K\}$ can be represented by a tuple $\langle\theta_k, \omega_k, \phi_k\rangle$. In this tuple, θ_k represents the real-time data generated by the user request task (such as sensor data generated in real-time by IoT devices, video data, etc.), ω_k represents the size of the application data of the task, and ϕ_k represents the necessary floating-point operations (FLOPs) required for task completion [23]. Let $\Delta = \{1, 2, ..., N\}$ represent the set of users governed by the server. Furthermore, the set $U^t_{N_d} = \{\mu^t_1, \mu^t_2, ..., \mu^t_N\}$ denotes the task requests of each user within the server d's range at time t. At the beginning of each time slot, MUs can either execute a new computation task different from μ^{t-1}_n, maintain the request from the previous time slot, or not request any service. Additionally, any user's request satisfies $\mu^t_n \in \{\mathrm{K} \cup \{0\}\}$.

The set $\mathrm{P}^{d,t} = \{p^{d,t}_1, p^{d,t}_2, ..., p^{d,t}_K\}$ is utilized to denote the application data retained in the cache pool of the MEC server node $d \in \Delta$ at time t. The element in the set satisfies $p^{d,t}_k \in \{0, 1\}$; 0 and 1, respectively, represent the absence or presence of application data k

in the cache pool of server node d. Meanwhile, the set $P_{d,t} = \{\rho_1^{d,t}, \rho_2^{d,t}, ..., \rho_K^{d,t}\}$ represents the cache replacement strategy. Each $\rho_k^{d,t}$ can assume one of three values: -1, 0, or 1, corresponding to the deletion, maintenance, or addition operation. Based on this, the cache replacement operation for application k on server d should adhere to

$$p_k^{d,t} = p_k^{d,t-1} + \rho_k^{d,t-1}, \forall p_k^{d,t} \geq 0. \tag{1}$$

The inequality $\forall p_k^{d,t} \geq 0$ represents the MEC server unable to delete data that do not exist in their cache pool. Further, considering the cache pool capacity S of server d, the system's cache status and replacement decisions at any given moment must subject to

$$\sum_{k=1}^{K} \omega_k (p_k^{d,t} + \rho_k^{d,t}) \leq \mathrm{S}, \forall t \in T, \forall d \in \Delta. \tag{2}$$

This ensures the integrity and efficiency of data management in the DESCO system.

3.2. Communication Model

MEC servers are homogeneous devices with the same computing, communication, and storage capabilities [24,25]. The available bandwidth of server d is bHz, with F orthogonal wireless subchannels. The bandwidth of subchannels $f \in F$ is $b = B/F$, where $\forall f \in \{1, 2, ..., F\}$. Each user can only occupy one channel to communicate with the server. The set $V^{d,t} = \{v_1^{d,t}, v_2^{d,t}, ..., v_N^{d,t}\}$ represents a server's wireless channels' allocation strategy while providing RCO service, and elements subjects to $\forall v_n^{d,t} \in \{0, 1, 2, ..., F\}$, where 0 signifies that the user processes computational data on their local device. However, users will generate transmission energy consumption, which is determined by two variables: the volume of transmitted data and the uplink transmission rate. The uplink transmission rate is influenced by the number of users occupying the channel $Q \in [0, N]$, user's channel gain ζ_n, and the transmission power ψ_n. According to the Shannon–Hartley theorem, the minimum effective uplink transmission rate for mobile user n managed by server d in time slot t is [26]

$$l_n^{d,t} = b\log_2\left(1 + \frac{\zeta_n \psi_n}{\sum\limits_{m \in \{Q-n\}, v_m^{d,t} = v_n^{d,t}} \zeta_m \psi_m + \sigma^2}\right), \tag{3}$$

where σ^2 represents the second central moment of Gaussian white noise. It is observed that the effective transmission power of user n decreases as the number of users Q occupying the same channel increases. Too many users occupying the same channel will increase noise interference $\sum_{m \in \{Q-n\}, v_m^{d,t} = v_n^{d,t}} \zeta_m \psi_m + \sigma^2$, resulting in a decrease in energy consumption benefits for task offloading, reducing the energy efficiency of task offloading and increasing task latency. Therefore, the impact of channel allocation must be considered in task offloading strategies to optimize performance.

3.3. Cache Chord Mechanism

In this section, we present a decentralized real-time resource access mechanism by orchestrating the network resources of MEC server clusters in the edge layer, utilizing a DHT-based consistent hashing ring network topology. Our discussion focuses on three key aspects: mapping, indexing, and updating for the Cache Chord.

3.3.1. Identifier Mapping

The initial step involves performing a consistent hashing operation on the NID of server nodes, and KID of application resource identifiers to construct a ring topology. The Hash algorithm maps these identifier data into a space of size 2^m, creating a unique m-bit string. The data in this space are arranged in ascending order, proceeding clockwise, to form

a virtual circular "ring". This ring is numerically labeled with integers within the range of $[0, 2^m - 1]$, assigning numbers to both the server and application data, which follows

$$X = \text{Hash}\langle \text{NID} \rangle \bmod m, \tag{4}$$

$$Y = \text{Hash}\langle \text{KID} \rangle \bmod m, \tag{5}$$

where Hash$\langle\rangle$ is the mapping function. Both random variables X and Y are defined as positive integers within interval $[0, 2^m - 1]$. The position of server node within the ring is denoted by Nod$X$, while the location of application node is represented by Key$Y$.

In the ring network, resource Key$Y$ is updated by the first server node Nod$X$ encountered in the clockwise direction. Nod$X$ maintains a cache IP list for resource localization. Specifically, the list comprises a dataset formed by tuple $\langle \text{Key}Y, \text{ServerIP}_Y \rangle$, the first element of this tuple represents the position Y of the application data within the ring, the IP address of each physical device that stores application data Y at time t is reperensted by

$$\text{ServerIP}_Y^t = \{nid_1, nid_2, ..., nid_S\} \cup \{0\}, \forall nid_s \in \text{NID}, \tag{6}$$

where $\{0\}$ signifies that the application data do not exist in edge layer. Utilizing this table, any user requesting service k for application data ω_k can determine the resource's position in the ring network by processing its resource identifier kid_k through Formula (5), and index which machine stored the application data through ServerIP_Y^t. It should be clarified that the positions and adjacent relationships of nodes within the Cache Chord merely represent a logical topology at the overlay network level, rather than actual storage locations. Additionally, the cache IP list on the server stores only indices of certain application data, not the application data themselves within the cache pool.

3.3.2. IP Indexing

In the ring topology, each server node Nod$X$ identifies the first server node encountered in the clockwise direction as its successor node, and the first server node encountered in the counterclockwise direction as its predecessor node. $\forall$Nod$X$ maintains the IP addresses of its successor and predecessor nodes, facilitating rapid access to the resources and cache IP lists of adjacent nodes.

Based on this structure, any server node d can determine the application's relative position based on the hash value of the corresponding identifier kid_k. By sequentially querying other server nodes in the order of the ring, the corresponding resource can be found in the respective successor node. However, the complexity of this query algorithm is not optimal. Assuming there are D servers in the ring, the query complexity is $O(D)$, and in the worst case, all nodes need to be traversed to find the target element.

To accelerate search processes, Cache Chord introduces the finger table mechanism [27]. Specifically, each server node Nod$X$ maintains a table consisting of successor nodes, each with a length of m. This list includes the IP addresses corresponding to these nodes. The sequence of nodes in this table is subject to Nod$\{(X + 2^{i-1}) \bmod 2^m\}$, where $i \in \{1, 2, ..., m\}$. However, if the i-th server node does not exist in the ring, the node closest to the i-th position is saved as the i-th entry in the finger table, denoted as FingerTable(i). When Nod$X$ seeks to retrieve a resource node Key$Y$, it initially searches for the IP address of the resource among its stored successor and predecessor nodes. If the corresponding identifier Y for the task is not found, the server searches the finger table to locate the server node with a hash value greater than Y and that is closest to it. If the node with the highest number in the finger table is still less than Y, the process jumps to Nod$(X + 2^{m-1})$, which is $\text{FingerTable}_{\max}(i)$, and repeats the aforementioned steps until Y is found, as detailed in Algorithm 1.

The integration of the finger table and cache IP list mechanisms enables a binary search-like method in Cache Chords, effectively converging the time complexity of resource searches to $O(\log D)$, where D is the total number of server nodes. Comparing the complexity of the finger table-based Algorithm 1 with the previously mentioned exhaustive search,

according to L'Hôpital's rule, we can obtain $\lim_{D\to\infty} D^{-1}\log D = 0$. This implies that such a nonlinear search method can locate the corresponding resources with fewer execution cycles. Figure 3 shows a simulation of a Cache Chord of size 2^7, demonstrating the process of server node nid_d mapping to the Cache Chord and searching for application Key78.

Algorithm 1 Application Key Routing

Initialize Nod$X$.successor, Nod$X$.*CacheIPlist*, Nod$X$.*FingerTable*, length of finger table m
repeat
  Search in Nod$X$.*CacheIPlist* and Nod$X$.successor
  if Nod$X$.*FingerTable*(i) > Key$Y$ and maximum i **then**
    ship application search request to Nod$X$.*FingerTable*(i)
  else if Key$Y$ > Nod$X$.*FingerTable*(m) **then**
    ship request to Nod$X$.*FingerTable*(m)
    let Nod$X$.*FingerTable*(m) be new Nod$X$
  end if
until find Nod$X$ which stored Key$Y$ in its finger table
return Key$Y$ IP address to original server node Nod$X$

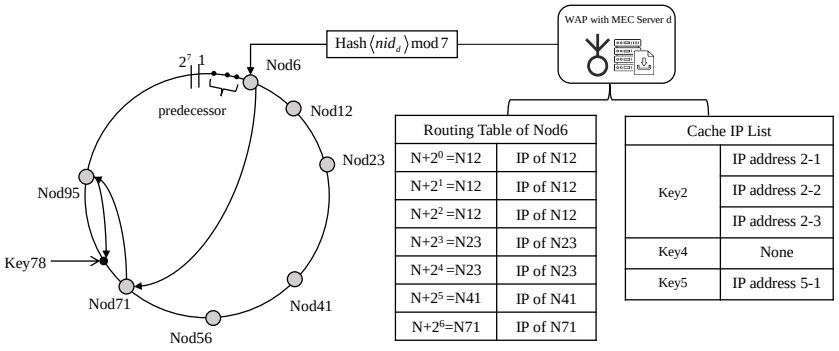

Figure 3. Cache Chord mechanism.

3.3.3. Identifier Mapping

At the end of time slot t, each server node in the edge layer executes strategy $æ_{d,t}$ to update the cache IP list. Specifically, if server d stores or deletes an application task in the local cache pool, i.e., $\rho_k^{d,t} = -1$ or 1, it requires executing Algorithm 2 to update the cache IP list. This ensures consistency between the data index in the overlay network and the storage status of data in the actual environment.

Algorithm 2 Cache IP List Update

Initialize NID, KID, Nod$X$.*CacheIPlist*, Nod$X$.*FingerTable*, cache replacement decision $æ_{d,t}$ of Nod$X$
for each server d in cluster Δ **do**
  for each task k in server d **do**
    if $\rho_k^{d,t} \neq 0$ **then**
      search Task k through execute **Algorithm 1** and get IP address of target server node Nod$T$
      if $\rho_k^{d,t} = 1$ **then**
        add Nod$X$ IP address into corresponding ServerIP$_Y^t$ of Nod$T$.*CacheIPlist*
      end if
    end if
  end for
end for
return Key$Y$ IP address to original server node Nod$X$

3.4. DESCO Task Processing Model

Algorithm 3 outlines the workflow of the mechanism. Initially, the server gathers individual user requests and searches its local cache pool $\mathbb{R}$ for corresponding application data ω_k. If the data are not already stored, the server then executes Algorithm 1 to locate the server node managing that application within the Cache Chord network by retrieving the cache IP list. If $\text{ServerIP}_Y^t \neq 0$, the IP of the node storing the application is returned to server d. Subsequently, the server derives the RCO strategy $\text{V}^{d,t}$ based on the local cache status and Cache Chord information. In the final phase, Algorithm 2 updates the resource index within the Cache Chord network. The output is the user layer energy consumption at that moment, which comprises computational and communication energy costs.

Algorithm 3 DESCO Network Overall Mechanism

Initialize MEC server cluster Δ, Cache Chord $\mathbb{C}$, MEC server local cache buffer $\mathbb{R}$
for each server d in cluster Δ **do**
  server d gathering user request set $U_{\text{N}_d}^t$
  for each user request μ_n^t **do**
    search corresponding application data ω_k in cache buffer $\mathbb{R}$
    if $p_k^{d,t} = 0$ **then**
      play **Algorithm 1** in Cache Chord $\mathbb{C}$ to search application data of task k
      server d compute offloading strategy $\text{V}^{d,t}$ based on application caching state and Cache Chord by playing **RCO decision**
    end if
    for each offloading vector $v_n^{d,t}$ **do**
      if $v_n^{d,t} \neq 0$ **and** $\text{ServerIP}_k \vee p_k^{d,t} = 0$ **then**
        user transmit task data ω_k and task data θ_k to server d, execute task on MEC server
      else if $v_n^{d,t} \neq 0$ **and** $\text{ServerIP}_k \vee p_k^{d,t} = 0$ **then**
        user transmit task data θ_k to server d, execute task on MEC server
      else
        local compute
      end if
      user n generated energy consumption $\text{Y}_{n,d,k,t}$
    end for
  end for
  update application state in Cache Chord through **Algorithm 2**
end for
return user layer energy consumption $\sum_{d \in \Delta} \sum_{n \in \text{N}_d, k \in U_{\text{N}_d}^t} \text{Y}_{n,d,k,t}$

3.4.1. Local Computing Mode

We derive the pure computational cost $o_{d,n}^2 \phi_k \varsigma$ based on the mobile device computational model [14], where the parameter $o_{d,n}$ represents the performance of the mobile user's computing unit, measured in floating-point operations per second (FLOPS). This metric can be adjusted through power control technology DVFS [28]. Additionally, the coefficient ς correlates with the power consumption and is inherently linked to the hardware architecture of the mobile device. Considering the time constraint τ of task μ_n^t, we ascertain that when $o_{d,n} = \phi_k / \tau$, the MUs can execute services with minimal power consumption without breaching the user's latency constraints, represented by

$$\varsigma \frac{{\phi_k}^3}{\tau^2}. \tag{7}$$

Meanwhile, we can conclude that the latency of local computation is equal to the maximum value of the time slot, τ.

3.4.2. Pure Offloading Mode

Communication cost is determined by the transmitted data volume, channel coherence parameters, and the user's transmission power. Notably, when both the local cache pool of server and the Cache Chord lack the application data ω_k for task k, i.e., $\text{ServerIP}_k^t \vee p_k^{d,t} = 0$, the user should upload both the application data ω_k and the real-time computational data θ_k to the edge. Consequently, it can be inferred that the current energy consumption of the n is

$$\psi_n \frac{\omega_k + \theta_k}{l_n^{d,t}}, \tag{8}$$

where ψ_n represents the transmission power of a user. Furthermore, we can infer that the latency of pure offloading is

$$T_{n,k,d,t}^P = \frac{\phi_k}{\eta_d} + \frac{\omega_k + \theta_k}{l_n^{d,t}}, \tag{9}$$

where the rightmost term represents the transmission delay for user n when sending data of task k, and the middle term represents the computation delay for server d.

3.4.3. Cache-Based Offloading Mode

When user n requests a service that is stored locally in the cache pool of d, or when the service k is located in another server through executing Algorithm 1 (denoted as $\text{ServerIP}_k^t \vee p_k^{d,t} = 1$), MUs only need to transmit θ_k to server. The corresponding transmission cost is then quantified as

$$\psi_n \frac{\theta_k}{l_n^{d,t}}; \tag{10}$$

similarly, the latency in cache-based offloading can be expressed as

$$T_{n,k,d,t}^C = \text{sgn}(1 - p_k^{d,t}) T_{d,k,t}^R + \frac{\phi_k}{\eta_d} + \frac{\theta_k}{l_n^{d,t}}, \tag{11}$$

where signum function $\text{sgn}(x)$ be subjected to

$$\text{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0. \end{cases} \tag{12}$$

In Equation (12), $\text{sgn}(1 - p_k^{d,t}) T_{d,k,t}^R$ denotes the latency of executing the routing algorithm to acquire application data ω_k in the Cache Chord. In this state, users do not need to further transmit data ω_k for task k, thereby reducing some of the time overhead. To simplify the model, consistent with other studies [3,29,30], we assume that the computational capacity of MEC servers is much greater than that of MUs, i.e., $\eta_d \gg o_{d,n}$. Therefore, the latency caused by MEC servers performing computations can be eliminated.

During the process of MEC servers utilizing the Cache Chord to execute indexing algorithms, the data volume transmitted between MEC nodes is very small, containing only information such as the IP address of the resources and the hash value of the destination server node address. Additionally, as MEC nodes are geographically adjacent and communicate through ideal wired backhaul links, without any transmission bottlenecks, the propagation delay of MEC servers transmitting application data to each other can be neglected. Therefore, in our modeling, we assume that the delay of MEC servers retrieving and transmitting corresponding application data ω_k in the Cache Chord can be ignored. This can be represented by the following inequality:

$$\frac{\phi_k}{\eta_d} + T_{d,k,t}^R \ll \frac{\phi_k}{o_{d,n}} \leq \tau \tag{13}$$

3.4.4. User Layer Energy Consumption

Let $\mathbb{E} = \{\varepsilon_1^t, \varepsilon_2^t, ..., \varepsilon_K^t\}, \forall \varepsilon_k^t \in \{0,1\}$ represent whether the edge layer holds application data ω_k, when the application data exist, the formula $\text{ServerIP}_k^t \vee p_k^{d,t} = 1$, $\varepsilon_k^t = 1$ can be derived; otherwise, $\text{ServerIP}_k^t \vee p_k^{d,t} = 0$, and $\varepsilon_k^t = 0$. Based on these, it can be concluded that at any slot t, the service request $\mu_n^t \in \{0,1,2,...,K\}$ of local user n on server d satisfies the following energy consumption formula:

$$\begin{aligned} \mathrm{Y}_{n,d,k,t} = &\operatorname{sgn}(\mu_n^t)\{(1-\operatorname{sgn}(v_n^{d,t}))\varsigma\frac{{\phi_k}^3}{\tau^2} \\ &+\operatorname{sgn}(v_n^{d,t})((1-\varepsilon_k^t)\psi_n\frac{\omega_k+\theta_k}{l_n^{d,t}}+\varepsilon_k^t\psi_n\frac{\theta_k}{l_n^{d,t}})\}, \end{aligned} \tag{14}$$

3.5. Problem Formulation

Based on the above modeling, the optimization objective can be transformed as follows. Under the assistance of the Cache Chord mechanism, take the RCO action $\mathrm{V}^{d,t}$ in order to minimize the long-term average energy consumption of the user layer in the DESCO model, which can be formulated as the following problem:

$$\mathrm{P}: \min_{\mathbb{V}^t} \left(\lim_{T\to\infty}\frac{1}{TD}\sum_{t=1}^{T}\sum_{d=1}^{D}\sum\nolimits_{n\in \mathrm{N}_d, k\in U_{\mathrm{N}_d}^t}\mathrm{Y}_{n,d,k,t}\right) \tag{15}$$

$$\text{s.t.} \quad nid_s \in \mathrm{NID}, \forall s \in S, \tag{16}$$

$$p_k^{d,t} = p_k^{d,t-1} + \rho_k^{d,t-1}, \forall p_k^{d,t} \geq 0, \forall t = T, \forall d \in \Delta, \tag{17}$$

$$\sum\nolimits_{k\in[1,K]}\omega_k(p_k^{d,t}+\rho_k^{d,t}) \leq \mathrm{S}, \forall t \in T, \forall d \in \Delta, \forall k \in \mathrm{K}, \tag{18}$$

$$\rho_k^{d,t} \in \{1,0,-1\}, \forall k \in U_{\mathrm{N}_d}^t, \forall t \in T, \forall d \in \Delta, \tag{19}$$

$$v_n^{d,t} \in \{0,1,2,...,F\}, \forall n \in \mathrm{N}_d, \forall t \in T, \forall d \in \Delta. \tag{20}$$

$$\operatorname{sgn}(v_n^{d,t})\left(\operatorname{sgn}(\varepsilon_k^t)T_{n,k,d,t}^C + \operatorname{sgn}(1-\varepsilon_k^t)T_{n,k,d,t}^P\right) \leq \tau. \tag{21}$$

Set $\mathbb{V}^t = \{\mathrm{V}^{1,t}, \mathrm{V}^{2,t}, ..., \mathrm{V}^{D,t}\}$ represents the entire server cluster of RCO decisions. (16) mandates that the storage location for any application data must reside within the servers at the edge layer. (17) specifies that the application cache status $p_k^{d,t}$ of MEC server at time t is determined by its cache status at $(t-1)$ and the application replacement decisions $\rho_k^{d,t-1}$ based on that status. (18) asserts that for any MEC server d, the current tasks and those newly added at the next moment, based on the application replacement decision, must not exceed the server's own cache pool capacity S. (19) illustrates the cache replacement policy of pool $\mathrm{P}^{d,t}$: addition, deletion, or retention. (20) represents the offloading decision for server d towards user n at time t, where $\{1,2,...,F\}$ denotes data offloading through the server's subchannel, and 0 indicates local computation of the task. The next section will detail how to decouple the complex problem P and address it through a joint algorithmic approach. (21) represents the latency constraint of real-time computing tasks.

4. Distributed Real-Time Computation Offloading Algorithm Based on MDPSO

4.1. Modeling the RCO Problem Based on Multi-Player Static Game

The coherent noise of a channel increases as the number of users occupying the channel rises. Consequently, users require more energy to transmit real-time sensor data, as indicated by Formula (3). Consequently, the offloading benefits of MUs selecting that channel decrease. Based on the above analysis, it can be observed that there exists an evident competitive relationship among MUs in the process of offloading data (selecting offloading channels). Moreover, user requests in any time slot occur concurrently rather than sequentially, which introduces the RCO problem into the realm of multi-player static game theory. In this chapter,

discussions will be conducted on this game problem, and the NE solution will be sought based on the multi-dimensional particle swarm method in heuristic algorithms.

In the computation offloading problem, as the popularity of tasks does not change due to users being in different coverage areas of MEC, the requests μ_n^t of any MUs in the terminal layer follow the same probability distribution. Additionally, the RCO decision of any MEC server only affects the data upload efficiency and latency of local users. Therefore, the scope of the computation offloading problem can be atomized, with each server making decisions assisted by the cache ring. This can be represented by each server minimizing its real-time energy consumption function $q_{d,t}(V^{d,t})$. Thus, the optimization problem becomes

$$\begin{aligned} \text{P1: } & \min_{\mathrm{V}^{d,t}} q_{d,t}(\mathrm{V}^{d,t}) = \sum\nolimits_{n\in \mathrm{N}_d, k\in U^t_{\mathrm{N}_d}} \mathrm{Y}_{n,d,k,t} \\ & \text{s.t.} \quad (17), (20), (21) \end{aligned} \tag{22}$$

where $\mathrm{V}^{d,t}$ represents an N-dimensional discrete random variable, and each dimension corresponds to the RCO strategy $v_n^{d,t}$ of user n. The effectiveness of a decision is quantified by the difference between the user's computational energy consumption and communication energy consumption. The offloading benefit for user $n \in \mathrm{N}_d$ is defined as

$$\Gamma_n(v_n^{d,t}) = \mathrm{sgn}(v_n^{d,t})(\varsigma \frac{{\phi_k}^3}{\tau^2} - \psi_n \frac{\theta_k + (1-\varepsilon_k^t)\omega_k}{b\log(1+\frac{\zeta_n \psi_n}{\sum_{m\in\{Q-n\}, v_m^{d,t}=v_n^{d,t}} \zeta_m \psi_m + \sigma^2})}), \tag{23}$$

where $\Gamma_n(v_n^{d,t}) > 0$ represents that implementing the offloading strategy $v_n^{d,t}$ can conserve energy for MUs in comparison to local computations. It is evident that the total offloading benefit $\sum_{n\in \mathrm{N}_d} \Gamma_n(v_n^{d,t})$ for the local user cluster of server d is inversely proportional to the real-time energy consumption function $q_{d,t}(\mathrm{V}^{d,t})$, which implies that

$$q_{d,t}(\mathrm{V}^{d,t}) \propto 1 \Big/ \sum\nolimits_{n\in \mathrm{N}_d} \Gamma_n(v_n^{d,t}). \tag{24}$$

In further analysis, we deduce that for user $\forall n \in \mathrm{N}_d$, the selection of channel $f \in F$ for transmission introduces coherent noise $\sum_{m\in\{Q-n\}, v_m^{d,t}=v_n^{d,t}} \zeta_m \psi_m + \sigma^2$ impacting both the individual benefit $\Gamma_n(v_n^{d,t})$ and the overall energy consumption $q_{d,t}(\mathrm{V}^{d,t})$.

This suggests that the RCO strategy $v_n^{d,t}$ of user n, in conjunction with other users' RCO strategies $\mathrm{V}_{n-}^{d,t} = \mathrm{V}^{d,t} - \{v_n^{d,t}\}$, formed a multi-player static game problem [31]. Consequently, this game can be characterized as

$$\mathcal{G} = \left\langle \mathrm{N}_d, \{\mathrm{P}_{n,t}\}, \Gamma_n(v_n^{d,t}) \right\rangle, \tag{25}$$

where N_d represents the set of players, $\{\mathrm{P}_{n,t}\}$ symbolizes the pure strategy set of player n, and $\Gamma_n(v_n^{d,t})$ denotes the offloading payoff compared to local computing. However, it is known that not all finite-strategy static games have pure-strategy Nash equilibrium. Therefore, we will analyze the existence of Nash equilibrium solutions for the given N-player static game problem $\mathcal{G}$.

Firstly, the number of players participating in this finite-strategy static game $\mathcal{G}$ is denoted as N. Let the n-th player's pure channel strategy available at time t be represented as

$$\mathrm{P}_{n,t} = \{A_1^n, A_2^n, ..., A_F^n\}, n \in \mathrm{N}_d. \tag{26}$$

The strategy space composed of pure strategies for N_d players is

$$\mathrm{P}_t = \mathrm{P}_{1,t} \times \mathrm{P}_{2,t} \times ... \times \mathrm{P}_{N,t}. \tag{27}$$

Any strategy combination in strategy space A is

$$p_t = \left(\mathrm{A}^1_{j_1,t}, \mathrm{A}^2_{j_2,t}, ..., \mathrm{A}^N_{j_N,t}\right), \mathrm{A}^n_{j_n,t} \in \mathrm{P}_{n,t}, n = 1, 2, ..., N. \tag{28}$$

Denote

$$\tilde{\mathrm{P}}^{n-1}_t = \mathrm{P}_{1,t}, \mathrm{P}_{2,t}, ..., \mathrm{P}_{n-1,t} (1 < n \leq N), \tilde{\mathrm{P}}^{n-1}_t = \Phi(n = 1), \tag{29}$$

$$\bar{\mathrm{P}}^{n+1}_t = \mathrm{P}_{n+1,t}, \mathrm{P}_{n+2,t}, ..., \mathrm{P}_{N,t} (1 \leq n < N), \bar{\mathrm{P}}^{n+1}_t = \Phi(n = N). \tag{30}$$

Apparently

$$\mathrm{P}_t = \begin{cases} \mathrm{P}_{1,t} \times \bar{\mathrm{P}}^{n+1}_t, & n = 1 \\ \tilde{\mathrm{P}}^{n-1}_t \times \mathrm{P}_{n,t}, & n = N \\ \tilde{\mathrm{P}}^{n-1}_t \times \mathrm{P}_{n,t} \times \bar{\mathrm{P}}^{n+1}_t, & 1 < n < N. \end{cases} \tag{31}$$

Without loss of generality, when $1 < n < N$, the following exists:

$$\begin{aligned} &\forall p^{n-1}_t \in \tilde{\mathrm{P}}^{n-1}_t, p^{n-1}_t = \left(\mathrm{A}^1_{j_1,t}, \mathrm{A}^2_{j_2,t}, ..., \mathrm{A}^{n-1}_{j_{n-1},t}\right), \mathrm{A}^k_{j_k,t} \in \mathrm{P}_{k,t}, k = 1, 2, ..., n-1, \\ &\forall p^{n-1}_t \in \bar{\mathrm{P}}^{n+1}_t, p^{n+1}_t = \left(\mathrm{A}^1_{j_{n+1},t}, \mathrm{A}^2_{j_{n+2},t}, ..., \mathrm{A}^N_{j_N,t}\right), \mathrm{A}^k_{j_k,t} \in \mathrm{P}_{k,t}, k = n+1, n+2, ..., N. \end{aligned} \tag{32}$$

According to the utility function $\Gamma_n(v^{d,t}_n)$ of the n-th player, if there exists the following:

$$p^*_t = \left(p^{n-1*}_t, \mathrm{A}^{n*}_{j_n,t}, p^{n+1*}_t\right), \tag{33}$$

let

$$\Gamma_n(p^*_t) = \Gamma_n\left(p^{n-1*}_t, \mathrm{A}^{n*}_{j_n,t}, p^{n+1*}_t\right) \geqslant \Gamma_n\left(p^{n-1*}_t, \mathrm{A}^n_{k,t}, p^{n+1*}_t\right), \forall \mathrm{A}^n_{k,t} \in \mathrm{P}_{n,t}. \tag{34}$$

Then, p^*_t is referred to as a pure-strategy Nash equilibrium of the finite-strategy static game $\mathcal{G}$. Building upon the above, we can continue the derivation; let

$$\begin{aligned} &T_{i,t}(p^{n-1}_t, p^{n+1}_t) = \left\{(p^{n-1}_t, A^n_{j_k,t}, p^{n+1}_t) | \max_{A^n_k \in \mathrm{P}_{n,t}} U_i(p^{n-1}_t, A^n_{k,t}, p^{n+1}_t)\right\}, i = 1, 2, \cdots, N \\ &T_{i,t} = \bigcup_{(p^{n-1}_t, p^{n+1}_t) \in \tilde{\mathrm{P}}^{n-1}_t \times \bar{\mathrm{P}}^{n+1}_t} T_i(p^{n-1}_t, p^{n+1}_t) = \\ &\{(p^{n-1}_t, A^n_{j_k,t}, p^{n+1}_t) | \max_{A^n_k \in \mathrm{P}_{n,t}} U_i(p^{n-1}_t, A^n_k, p^{n+1}_t)\}, \\ &\forall (p^{n-1}_t, p^{n+1}_t) \in \tilde{\mathrm{P}}^{n-1}_t \times \bar{\mathrm{P}}^{n+1}_t | \ i = 1, 2, \cdots, N. \end{aligned} \tag{35}$$

If the game $\mathcal{G}$ has a Nash equilibrium, then

$$p_t = \left(p^{n-1}_t, \mathrm{A}^n_{j_n,t}, p^{n+1}_t\right) = \left(\mathrm{A}^1_{j_1,t}, \mathrm{A}^2_{j_2,t}, ..., \mathrm{A}^N_{j_N,t}\right). \tag{36}$$

According to the definition of pure NE, we have

$$\Gamma_i(p^{n-1}_t, A^n_{j_n,t}, p^{n+1}_t) \geqslant \Gamma_i(p^{n-1}_t, A^n_{k,t}, p^{n+1}_t), \forall A^n_{k,t} \in \mathrm{P}_{n,t}. \tag{37}$$

Therefore, there exists:

$$(p^{n-1}_t, A^n_{j_n,t}, p^{n+1}_t) \in T_n\left(p^{n-1}_t, p^{n+1}_t\right) \subset T_n, n = 1, 2, ..., N. \tag{38}$$

Thereby

$$(p^{n-1}_t, A^n_{j_n,t}, p^{n+1}_t) \in \bigcap_{i=1}^{N} T_{i,t}. \tag{39}$$

So, we have

$$\bigcap_{i=1}^{N} T_{i,t} \neq \Phi. \tag{40}$$

If the above equation holds, then we may assume the following:

$$p_t = \left(p_t^{n-1}, \mathrm{A}_{j_n,t}^{n}, p_t^{n+1}\right) = \left(\mathrm{A}_{j_1,t}^{1}, \mathrm{A}_{j_2,t}^{2}, ..., \mathrm{A}_{j_N,t}^{N}\right) \in T_{i,t}, i = 1, 2, ..., N. \tag{41}$$

From the definition of $T_{i,t}$, we can infer that

$$\begin{aligned} \Gamma_n\left(\mathrm{A}_{j_1,t}^{1}, \mathrm{A}_{j_2,t}^{2}, ..., \mathrm{A}_{j_N,t}^{N}\right) &= \max_{A_k^n \in \mathrm{P}_{n,t}} \Gamma_n\left(p_t^{n-1}, A_{k,t}^{n}, p_t^{n+1}\right) \\ &\geq \Gamma_n\left(p_t^{n-1}, A_{k,t}^{n}, p_t^{n+1}\right), \forall A_{k,t}^{n} \in \mathrm{P}_{n,t}. \end{aligned} \tag{42}$$

According to the definition of pure-strategy Nash equilibrium, a pure-strategy Nash equilibrium of a finite-strategy static game is $\left(\mathrm{A}_{j_1,t}^{1}, \mathrm{A}_{j_2,t}^{2}, ..., \mathrm{A}_{j_N,t}^{N}\right)$. Based on the above derivation, it can be concluded that the sufficient and necessary condition for the existence of a pure-strategy Nash equilibrium solution is $\bigcap_{n=1}^{N} T_{n,t} \neq \Phi$.

Clearly, there exists a set of solution vectors for the game $\mathcal{G}$, such that

$$\Gamma_n(\mathrm{V}_{n-}^{d,t}, v_n^{d,t*}) \geq \Gamma_n(\mathrm{V}_{n-}^{d,t}, v_n^{d,t}), v_n^{d,t} \in \{0, 1, 2, ..., F\}. \tag{43}$$

Therefore, the Nash equilibrium solution for the game problem $\mathcal{G}$ exists.

4.2. Solving RCO Using Multi-Dimensional Discrete Particle Swarm Optimization Algorithm

To resolve this game problem, we employed a multi-dimensional discrete particle swarm algorithm [32] to search for NE point $\tilde{\mathrm{V}}^{d,t}$. The modeling process begins by initializing a set of multi-dimensional random particles, represented by the set $\mathcal{J}_{\mathrm{N}_d}^{l,i} = \{J_1^{l,i}, J_2^{l,i}, ..., J_N^{l,i}\}, \forall i \in I, \forall l \in L$. Here, i denotes the iteration rounds of the particles, L represents the population size, and the particle dimension corresponds to the total number of users Nd participating in the RCO game.

Each particle symbolizes a feasible solution set for the game, with the solution vector's domain in any dimension n following $\forall J_n^{l,i} \in \{0, 1, 2, ..., F\}$. It is important to note that for users without service requests at slot t, the corresponding dimensional value remains consistently zero, as indicated by the expression $\mu_n^t = 0 \rightarrow J_n^{l,i} \equiv 0$.

These particles are initially distributed randomly within the solution space and are assessed for estimated energy consumption $q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{l,i})$ based on the fitness function $q_{d,t}()$, which reflects the quality of the particle's current position. The solution vector with the best fitness in the global historical iterations for population L is designated as $\mathcal{J}_{\mathrm{N}_d}^{globe} = \{J_1^{globe}, J_2^{globe}, ..., J_N^{globe}\}$, while the local historical optimal fitness solution vector for particle $\forall l \in L$ is named by $\mathcal{J}_{\mathrm{N}_d}^{local} = \{J_1^{local}, J_2^{local}, ..., J_N^{local}\}$. In the i-th iteration, particles determine their displacement vector $\mathcal{Z}_{\mathrm{N}_d}^{l,i} = \{Z_1^{l,i}, Z_2^{l,i}, ..., Z_N^{l,i}\}$ for the subsequent iteration by considering the global optimal solution in the $(i-1)$-th iteration, local optimal solution, and the particle's current position, where positive integer $\forall Z_n^{l,i} \in [0, F]$. The formula for updating the displacement vector in the n-th dimension for the i-th iteration is as follows:

$$Z_n^{l,i} = Z_n^{l,i-1} + \left\lfloor H1(x_1) \times (J_n^{local} - J_n^{l,i-1}) \right\rfloor + \left\lfloor H2(x_2) \times (J_n^{globe} - J_n^{l,i-1}) \right\rfloor \tag{44}$$

In cases where the user request is null, the particle displacement vector should be zero. To increase the randomness of particle movement, we set two exploration coefficients, $H1(x_1)$ and $H2(x_2)$. Both of them follow uniform distribution. The random variables x_1

and x_2 have value ranges of $[h1_{\min}, h1_{\max}]$ and $[h2_{\min}, h2_{\max}]$, respectively, and satisfy the following equation:

$$H1(x_1) = \begin{cases} \frac{1}{(h1_{\max} - h1_{\min})}, & x_1 \in [h1_{\min}, h1_{\max}] \\ 0, & \text{elsewhere,} \end{cases}$$
$$H2(x_2) = \begin{cases} \frac{1}{(h2_{\max} - h2_{\min})}, & x_2 \in [h2_{\min}, h2_{\max}] \\ 0, & \text{elsewhere.} \end{cases} \tag{45}$$

Nevertheless, the exploration coefficient results in the weighted displacement vector fail to comply with the domain definition of the action space, specifically concerning decimal values. To mitigate the impact of the decimal component in our calculations, we employed the floor function $\lfloor * \rfloor$ to handle the increments. This approach guarantees that the displacement vector adheres to the constraints imposed by the offloading space. The iterative formula to determine the particle's position is delineated as follows:

$$\mathcal{J}_{\mathrm{N}_d}^{l,i} = \mathcal{J}_{\mathrm{N}_d}^{l,i-1} + \mathcal{Z}_{\mathrm{N}_d}^{l,i}. \tag{46}$$

After i iterations, the algorithm arrives at the final global optimal solution vector $\mathcal{J}_{\mathrm{N}_d}^{final}$, which minimizes the real-time energy consumption function $q_{d,t}^{\min}(\mathcal{J}_{\mathrm{N}_d}^{final})$. This solution vector also represents the NE point $\tilde{\mathrm{V}}^{d,t}$ of the game problem $\mathcal{G}$. The specific process of the RCO decision is detailed in Algorithm 4.

Algorithm 4 Discrete Multi-dimensional PSO-based RCO Algorithm

Initialize: MEC server d, Cache Chord $\mathbb{C}$, MEC server local cache buffer $\mathbb{R}$, user request set $U_{\mathrm{N}_d}^t$, communication channel F, game $\mathcal{G}$, particle population quantity L, particle iteration i, fitness function $q_{d,t}()$, particle displacement vector $\mathcal{Z}_{\mathrm{N}_d}^{l,i}$, particle $\mathcal{J}_{\mathrm{N}_d}^{l,i}$
for each iteration i **do**
  for each particle $\mathcal{J}_{\mathrm{N}_d}^{l,i}$ **do**
    for each dimension n **do**
      calculate displacement vector $\mathcal{Z}_{\mathrm{N}_d}^{l,i}$ through **function** (44)
      update particle position $\mathcal{J}_{\mathrm{N}_d}^{l,i}$ through **function** (46)
    end for
    calculate fitness $q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{l,i})$
  end for
  if $q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{l,i}) > q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{local})$ **then**
    $\mathcal{J}_{\mathrm{N}_d}^{local} = \mathcal{J}_{\mathrm{N}_d}^{l,i}$
  end if
  if $q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{l,i}) > q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{local})$ **and** $\forall q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{\xi,i}) < q_{d,t}(\mathcal{J}_{\mathrm{N}_d}^{l,i}), \forall \xi \in L$ **then**
    $\mathcal{J}_{\mathrm{N}_d}^{globe} = \mathcal{J}_{\mathrm{N}_d}^{l,i}$
  end if
end for
$\mathcal{J}_{\mathrm{N}_d}^{final} = \mathcal{J}_{\mathrm{N}_d}^{globe}$
return Global Optimal Solution $\mathcal{J}_{\mathrm{N}_d}^{final}$ to server d

5. Performance Evaluation

Simulation experiments were developed on the Python platform for the DESCO network environment. The MDPSO algorithm was deployed based on this distributed environment. The service coverage area of distributed MEC nodes is a regular hexagonal region with a diagonal length of 200 m. The values of system time slots, system operating cycle, server's wireless transmission bandwidth, server's and MUs' computing power, user's channel gain ζ_n, and the local computing energy consumption coefficient ς are all

constants and remain unchanged in the subsequent comparative experiments. The specific values are shown in Table 2.

Table 2. Simulation setting.

Parameter	Value
Number of MEC servers Δ	3
Subchannel bandwidth b	3
ς	5×10^{-27}
σ^2	2×10^{-13}
ψ_n	0.5 W
F	10
D	2000
$o_{d,n}$	1 GHz
$\theta_{\max}, \omega_{\max}, \phi_{\max}$	5
$[h1_{\min}, h1_{\max}]$ and $[h2_{\min}, h2_{\max}]$	$[0, 3]$
Particle number L	500
Particle iteration i	100

During the initialization phase, DESCO randomly generates the data for all services in the set K. The values are sampled from the intervals $\theta_k \in [1, \theta_{\max}]$, $\omega_k \in [1, \omega_{\max}]$, and $\phi_k \in [1, \phi_{\max}]$, where the maximum values $\phi_{\max}, \omega_{\max}, \theta_{\max}$ are all set to 5. The service cache capacity S of MEC server d is fixed at 2GB, and the cache replacement policy is set to random caching by default. In the subsequent experiments, we will investigate the average energy consumption of users under different numbers of MEC servers. The environment configuration is the same as [3,31,33]. The state transition probabilities of user requests follow a Zipf distribution parameterized by $\langle R, L, \vartheta \rangle$, where

$$\Pr\{\mu_n^{t+1} = b | \mu_n^t = a\} = \begin{cases} R, \mathrm{a} \neq 0, b = 0 \\ \frac{1-R}{L}, a \neq 0, b = (a+l) \bmod (K+1), l \in [1, K] \\ \frac{1-R}{b^{\vartheta} \sum_{i \in [1,K]} (\frac{1}{i})^{\vartheta}}, a = 0, b \neq 0, \end{cases} \tag{47}$$

whereby ϑ represents the Zipf distribution parameter. L denotes the number of adjacent services that may be requested in the next stage. R signifies the probability that user n will not request any service in the subsequent phase. We will adjust parameters L and R to evaluate the performance of the algorithm under varying transition probabilities.

(1) The Greedy–Random algorithm: combines Greedy RCO with random cache replacement strategy. In the initialization phase, MEC servers randomly generate the RCO strategies for the user. Then, based on this initial strategy, the algorithm traverses each user's offloading decision in the set N_d. At each step, it searches for the channel occupancy strategy that minimizes the energy consumption for that user. Meanwhile, the MEC server adopts a random cache replacement strategy for local cache space. This strategy randomly replaces the stored application data with the application data offloaded by users in the current time slot after the cache space becomes saturated, satisfying Equation (18) during this process.

(2) Random cache replacement with multi-dimensional discrete particle swarm offloading (MDPSO-Random): The MDPSO strategy is utilized to solve RCO strategy. By assigning initial momentum to random multi-dimensional particles, the RCO strategy explores the optimal strategy in the solution space based on the fitness function. It is worth noting that this algorithm is deployed only in DESCO networks with a single MEC server.

(3) The MDPSO-Random algorithm with Cache Chord (MDPSO-Random with CC) builds upon the MDPSO-Random algorithm: This mechanism allows MUs to access the application cache resources across the entire edge layer.

The effectiveness of the MDPSO algorithm and Cache Chord is verified from three aspects: convergence of the MDPSO algorithm, energy consumption performance, and cache hit rate.

Firstly, the convergence analysis of the proposed algorithm is conducted. Figure 4 illustrates the instantaneous energy consumption obtained after iterations for both the MDPSO algorithm and the Greedy algorithm in the single-server scenario. The experiment is conducted with different MU counts ($N_d = 15, 25, 35, 45$) and a local MEC cache capacity of 2 GB. The horizontal axis represents the number of iterations for particles or algorithms, while the vertical axis represents the energy consumption of the MEC server. In the experiment, the dimension L of particles is set to 500, and the number of iterations i is 100.

The blue line represents the convergence curve of the MDPSO, while the orange line represents the Greedy algorithm. The green horizontal line, which remains constant throughout iterations, represents the energy consumption of local computation. Comparing the two algorithms, it is observed that both MDPSO and Greedy strategies converge within the first 40 rounds and can reduce user-level energy consumption by 18.75% to 41.17% compared to local computation. Moreover, MDPSO exhibits better convergence compared to the Greedy strategy, with an average reduction of 19.7% in user-level energy consumption per round. This indicates that the MDPSO algorithm can explore the solution space more comprehensively, avoiding local optima, and performs well when dealing with discrete vectors.

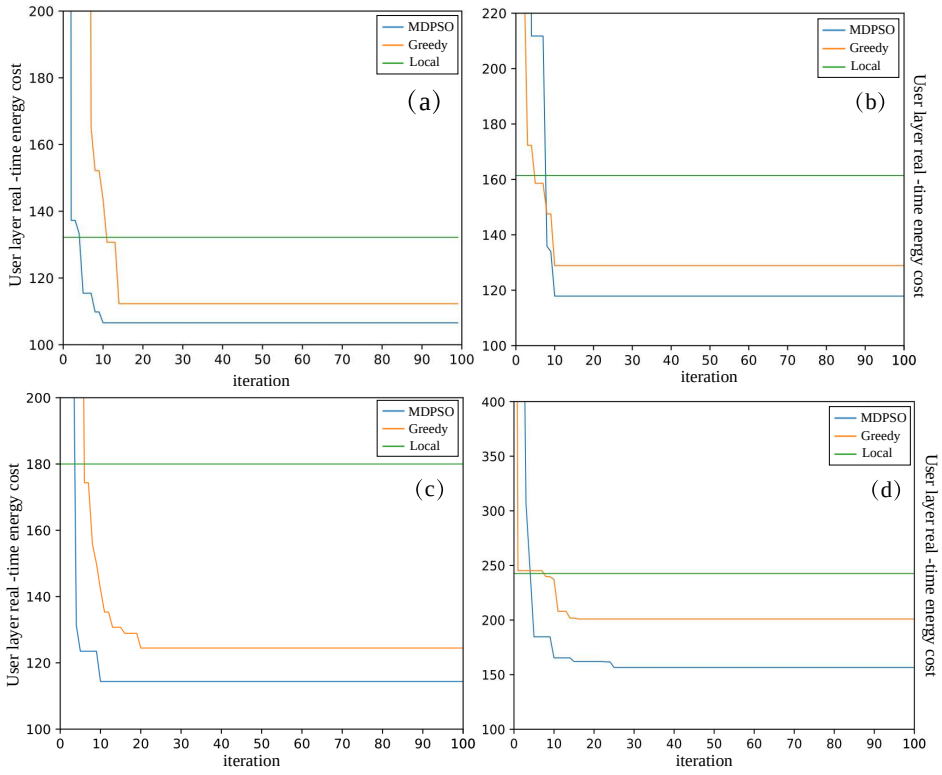

Figure 4. User layer real-time energy cost under different algorithms and different numbers of MUs: (**a**) MU number = 15. (**b**) MU number = 25. (**c**) MU number = 35. (**d**) MU number = 45.

Figure 5 illustrates the average task completion time of user layer per time slot with a task latency constraint of $\tau = 20$ ms. The red line represents the MDPSO algorithm with collaboration among five nodes, while the green line represents the Greedy offloading strategy with the same number of collaborating nodes. Since MUs can adopt DVFS to adjust the computational power of local devices and ensure timely task completion, the task completion time in each round remains stable at a maximum latency constraint of 20 ms. After running for 1000 time slots, the proposed algorithm outperforms others, with an average latency of 7.4 ms, while the Greedy strategy exhibits an average latency of

11.4 ms, both showing a decrease compared to the local computing. This implies that the proposed algorithm can converge to the optimal offloading strategy, thereby applying the cache-based offloading mode as much as possible, to reduce data transmission overhead for MUs.

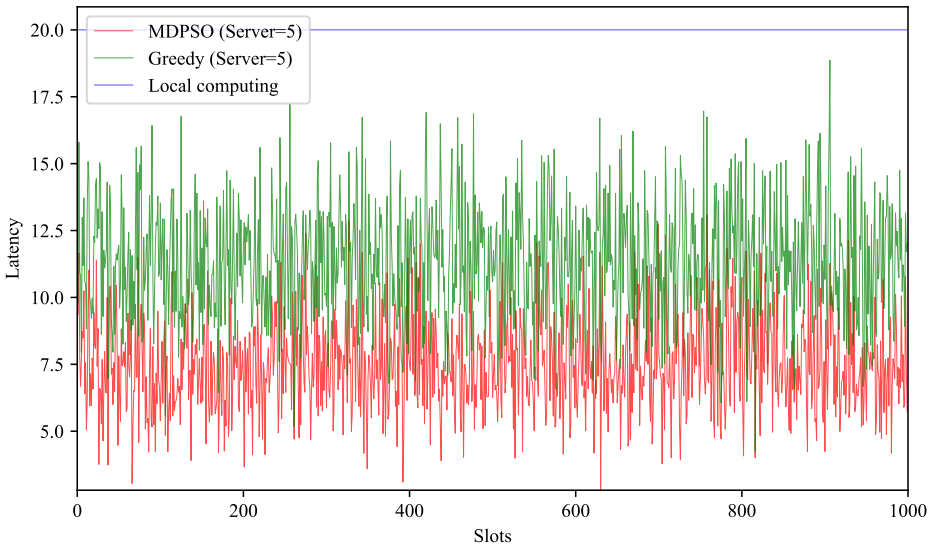

Figure 5. The comparison of average user layer latency per time slot. Models include MDPSO (server = 5), Greedy (server = 5), and local computing.

Figure 6 compares the energy consumption of local computation, Greedy-Random, MDPSO-Random, and MDPSO-Random with Cache Chord (CC)-assisted algorithms under different numbers of MUs and task richness conditions, using the default Zipf service request distribution function. In Figure 6a, the number of users covered by each MEC server is fixed at 5. Through the experiments, it is observed that, relative to local computation, the other three algorithms effectively reduce energy consumption overhead. In Figure 6a, the curve for local energy consumption remains relatively constant. This is because the energy consumption of local computation is only dependent on the number of tasks and FLOPs of tasks. The increase in the number of user types only poses challenges for cache-based algorithms. Moreover, through horizontal comparison, it is noted that the algorithm based on the cooperation of 5 distributed MEC servers with cache exhibits the best performance. This is attributed to the CC data-sharing mechanism, which allows users at any location to access resources across the entire MEC edge layer. This implies that the cache space linearly increases with the addition of servers in the CC network, thereby eliminating the need for users to transmit θ_k and resulting in a higher transmission of energy consumption.

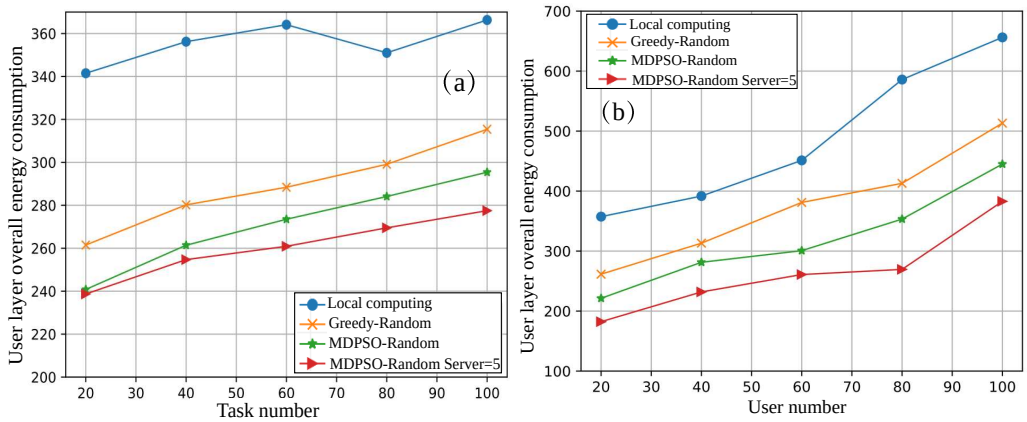

Figure 6. User layer overall energy cost comparison. (**a**) Task number comparison. (**b**) User number comparison.

Figure 7 compares the average energy consumption of a cluster of users under the collaboration of different numbers of MEC servers. As the number of MEC servers increases, the energy consumption in all cases shows a decrease and converges to the same value in all cases, because as the total number of server D increases, the cache pool storage is richer, which allows more requests to be converted from pure offloading to cache-based offloading until all MUs do not need to upload application data ω_k. It can also be noticed that the slopes of the three energy consumption curves decrease gradually as the total number of tasks increases. The energy consumption curve decreases most rapidly when the total number of tasks is small ($N_d = 100$) and converges at around 20 servers; the energy consumption curve of $N_d = 100$ converges at $D = 45$ because when the total number of tasks is larger, the servers also need to collaborate on a larger scale to make sure that the corresponding application data are stored.

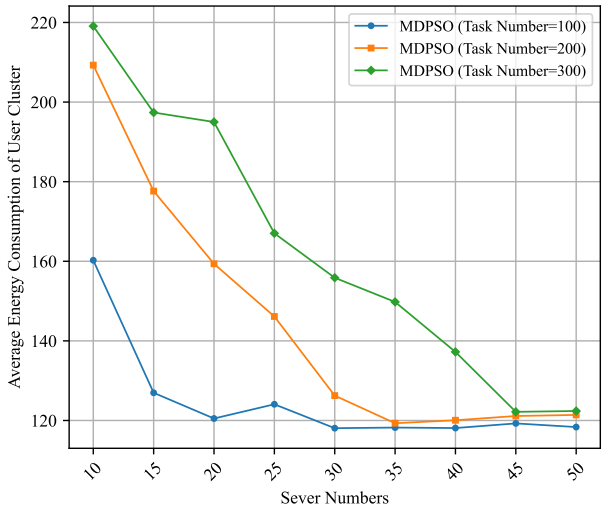

Figure 7. The comparison of average energy consumption under different numbers of MEC servers, where task number = 100, 200, 300.

Figure 8 illustrates the average energy consumption of MUs per time slot under different Zipf parameters and various task attributes for MDPSO. As depicted in Figure 8a, it can be observed that with the increase in the maximum value of application data ω_{max}, the average energy consumption of the Greedy algorithm and the MDPSO assisted by CC continues to rise. This is due to the increase in ω_{max}, which increases the transmission overhead of tasks, thereby reducing the offloading benefits. The proposed algorithm can effectively reduce the average energy consumption by 10.9% to 25.17% compared to the Greedy offloading strategy. Additionally, under the same parameter conditions ($L = 3$), MDPSO based on multi-server collaboration can reduce the average energy consumption by 7.89% to 4.7% compared to the single-node mode.

Observing Figure 8b, we can conclude that with the growth of θ_{max}, the average energy consumption of all algorithms increases. Moreover, comparing the Greedy algorithm with the MDPSO algorithm based on multi-node collaboration, under the same Zipf R condition, MDPSO can reduce the average energy consumption by 2.1% to 16.8% compared to Greedy. Furthermore, with the increase in parameter R, the energy consumption also decreases. This is because parameter R is positively correlated with the probability that the MUs' next stage service request is empty. A larger R means more MUs with empty requests at any time. Sparse service requests undoubtedly lead to reduced transmission energy consumption for MUs.

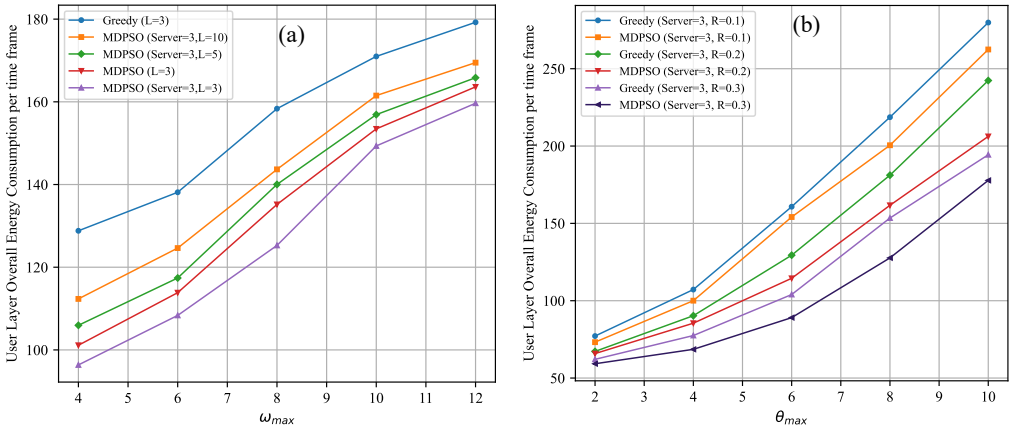

Figure 8. Average energy consumption under different Zipf parameters: (**a**) Zipf L; (**b**) Zipf R.

Figure 9a,b, respectively, explore the impact of the number of servers in the CC mechanism on the cache hit ratio from the perspectives of user count and service types. The cache hit ratio represents the ratio of the total number of accesses to the edge layer by users C_s to the number of services successfully requested C_h, expressed as the following formula:

$$\text{Cache hit ratio} = \frac{C_h}{C_s} \tag{48}$$

By vertically comparing Figure 9, it is evident that under the same conditions, as the number of MEC servers increases, the cache space grows linearly, leading to a significant improvement in the cache hit ratio C_r. Compared to a single server, when the number of servers is increased to 10, the cache hit ratio C_r increases by 33.3% to 42.7%. Moreover, when the number of users N_d is 20 and the number of services $K < 80$, the cache hit ratio achieved by the cooperation of 10 servers is consistently above 91%. This improvement is attributed to the Cache Chord mechanism overcoming the bottleneck imposed by the storage space limitation.

However, as the number of task types increases, the cache hit ratio decreases. This decline is due to the sparser distribution of user service requests as the task types increase. For individual MEC servers, the proportion of service data stored in the limited space of the local cache pool becomes relatively smaller compared to the overall request volume, leading to a decrease in C_r. Comparing the four models, it can be concluded that the Cache Chord mechanism effectively assists in computation offloading. Moreover, with the increase in node scale, higher benefits can be achieved.

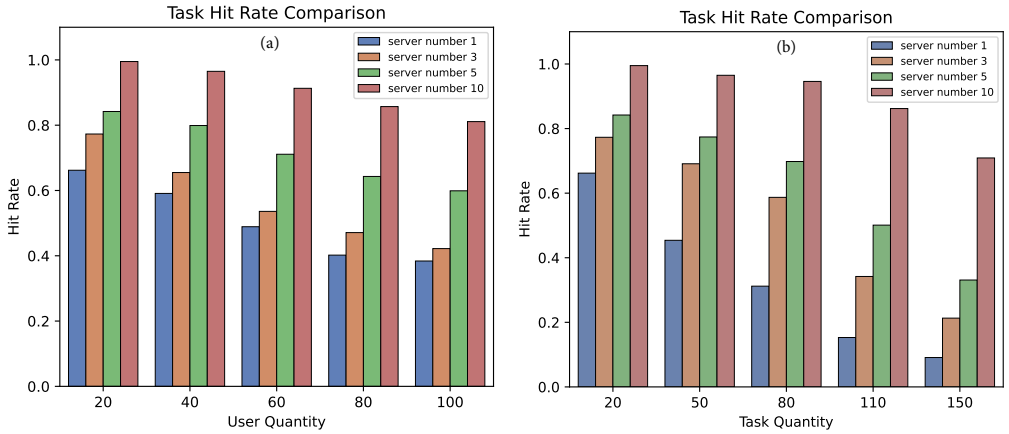

Figure 9. Cache hit rate under different server numbers in Cache Chord. (**a**) User quantity. (**b**) Task quantity.

6. Conclusions

This paper investigates how to provide higher-quality services to MUs while reducing user layer energy consumption in MEC networks. The DESCO network architecture is proposed to address this issue. In the DESCO network, a cache-sharing mechanism, Cache Chord, is conducted based on a geo-distributed edge server collaboration. Cache Chord utilizes consistent hashing to map servers and real-time applications into a circular logical topology. We also improved the IP routing algorithm to reduce computational complexity and designed an index update mechanism to ensure information synchronization among edge nodes. Furthermore, in addressing the bottleneck of channel allocation caused by concurrent requests from current users, the problem is formulated as a multi-player static game. After proving the existence of the Nash equilibrium point, the MDPSO algorithm is proposed to simulate user requests as particles, exploring the global optimal node in the solution space to provide users with the lowest energy consumption offloading strategy. Finally, the algorithm is compared with baseline algorithms, and the performance of the Cache Chord structure is tested at different scales.

Through simulation experiments, we have found that computation offloading assisted by the Cache Chord mechanism can significantly reduce user energy consumption overhead. It is foreseeable that with the increasing scale of added server nodes, energy consumption will be further reduced. However, this paper overlooks the fact that with the increase in server nodes, the transmission delay of the corresponding backhaul links will also increase, posing new challenges for performance optimization. Therefore, the next step will be to explore how larger node scales will affect service quality and consider deploying the protocol in a real network environment to test its performance.

Author Contributions: Conceptualization, W.L.; Data curation, W.L.; Formal analysis, Y.L., W.L. and S.L.; Funding acquisition, Z.Y.; Investigation, P.Z.; Methodology, Y.L., P.Z., Z.Y. and W.L.; Project administration, Y.L. and Z.Y.; Resources, S.L.; Software, P.Z. and S.L.; Validation, Y.L.; Visualization, Z.Y.; Writing—original draft, P.Z.; Writing—review and editing, Y.L. and P.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by Science and Technology Project of Hebei Education Department ZD2022102.

Data Availability Statement: The data can be shared upon request and are not publicly available due to the data also forms a part of an ongoing study.

Acknowledgments: The authors gratefully appreciate the anonymous reviewers for their valuable comments.

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

Abbreviations

The following abbreviations are used in this manuscript:

MEC	Multi-Access Edge Computing
WAP	Wireless Access Point
RSU	Road Side Unit
TDMA	Time-Division Multiple Access
DESCO	Distributed Edge Service Caching and Offloading
RCO	Real-Time Computation Offloading
NE	Nash Equilibrium
CC	Cache Chord
MUs	Mobile Users
DHT	Distributed Hash Table
P2P	Peer to Peer
DVFS	Dynamic Voltage and Frequency Scaling
FLOPs	Floating-Point of Operations
FLOPS	Floating-Point Operations Per Second
MDPSO	Multi-Dimensional Discrete Particle Swarm Optimization

References

1. Chen, X.; Cai, Y.; Li, L.; Zhao, M.; Champagne, B.; Hanzo, L. Energy-efficient resource allocation for latency-sensitive mobile edge computing. *IEEE Trans. Veh. Technol.* **2019**, *69*, 2246–2262. [CrossRef]
2. Lai, Z.; Liu, W.; Wu, Q.; Li, H.; Xu, J.; Wu, J. SpaceRTC: Unleashing the low-latency potential of mega-constellations for real-time communications. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications, Virtual Event, 2–5 May 2022 ; IEEE: Piscataway, NJ, USA, 2022; pp. 1339–1348.
3. Spinelli, F.; Mancuso, V. Toward enabled industrial verticals in 5G: A survey on MEC-based approaches to provisioning and flexibility. *IEEE Commun. Surv. Tutor.* **2020**, *23*, 596–630. [CrossRef]
4. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on multi-access edge computing for internet of things realization. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2961–2991. [CrossRef]
5. Mach, P.; Becvar, Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1628–1656. [CrossRef]
6. Yao, J.; Han, T.; Ansari, N. On mobile edge caching. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 2525–2553. [CrossRef]
7. Kang, J.; Yu, R.; Huang, X.; Wu, M.; Maharjan, S.; Xie, S.; Zhang, Y. Blockchain for secure and efficient data sharing in vehicular edge computing and networks. *IEEE Internet Things J.* **2018**, *6*, 4660–4670. [CrossRef]
8. Li, Q.; Zhang, Y.; Pandharipande, A.; Ge, X.; Zhang, J. D2D-assisted caching on truncated Zipf distribution. *IEEE Access* **2019**, *7*, 13411–13421. [CrossRef]
9. Lou, J.; Luo, H.; Tang, Z.; Jia, W.; Zhao, W. Efficient container assignment and layer sequencing in edge computing. *IEEE Trans. Serv. Comput.* **2022**, *16*, 1118–1131. [CrossRef]
10. Shi, Y.; Yang, K.; Jiang, T.; Zhang, J.; Letaief, K.B. Communication-efficient edge AI: Algorithms and systems. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2167–2191. [CrossRef]
11. Zhang, L.; Fu, D.; Liu, J.; Ngai, E.C.H.; Zhu, W. On energy-efficient offloading in mobile cloud for real-time video applications. *IEEE Trans. Circuits Syst. Video Technol.* **2016**, *27*, 170–181. [CrossRef]
12. Zhan, C.; Hu, H.; Sui, X.; Liu, Z.; Niyato, D. Completion time and energy optimization in the UAV-enabled mobile-edge computing system. *IEEE Internet Things J.* **2020**, *7*, 7808–7822. [CrossRef]
13. Yadav, R.; Zhang, W.; Elgendy, I.A.; Dong, G.; Shafiq, M.; Laghari, A.A.; Prakash, S. Smart healthcare: RL-based task offloading scheme for edge-enable sensor networks. *IEEE Sens. J.* **2021**, *21*, 24910–24918. [CrossRef]
14. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K.B. A survey on mobile edge computing: The communication perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358. [CrossRef]
15. Yu, S.; Langar, R.; Fu, X.; Wang, L.; Han, Z. Computation offloading with data caching enhancement for mobile edge computing. *IEEE Trans. Veh. Technol.* **2018**, *67*, 11098–11112. [CrossRef]
16. 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]
17. Tang, Q.; Fei, Z.; Li, B.; Han, Z. Computation offloading in LEO satellite networks with hybrid cloud and edge computing. *IEEE Internet Things J.* **2021**, *8*, 9164–9176. [CrossRef]
18. Mei, J.; Tong, Z.; Li, K.; Zhang, L.; Li, K. Energy-Efficient Heuristic Computation Offloading With Delay Constraints in Mobile Edge Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 4404–4417. [CrossRef]
19. Zorgati, H.; Djemaa, R.B.; Amous, I. Efficient IoT resource discovery approach based on P2P networks and Fog Computing. *Internet Things* **2023**, *24*, 100954. [CrossRef]
20. Wang, C.; Chen, C.; Pei, Q.; Lv, N.; Song, H. Popularity incentive caching for vehicular named data networking. *IEEE Trans. Intell. Transp. Syst.* **2020**, *23*, 3640–3653. [CrossRef]
21. He, Q.; Tan, S.; Chen, F.; Xu, X.; Qi, L.; Hei, X.; Jin, H.; Yang, Y. Edindex: Enabling fast data queries in edge storage systems. In Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, Taipei, Taiwan, 23–27 July 2023; pp. 675–685.
22. D'Angelo, M.; Caporuscio, M. Sa-chord: A self-adaptive p2p overlay network. In Proceedings of the 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W), Trento, Italy, 3–7 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 118–123.
23. Yu, F.; Cui, L.; Wang, P.; Han, C.; Huang, R.; Huang, X. Easiedge: A novel global deep neural networks pruning method for efficient edge computing. *IEEE Internet Things J.* **2020**, *8*, 1259–1271. [CrossRef]
24. Ning, H.; Li, Y.; Shi, F.; Yang, L.T. Heterogeneous edge computing open platforms and tools for internet of things. *Future Gener. Comput. Syst.* **2020**, *106*, 67–76. [CrossRef]
25. Jin, Y.; Cai, J.; Xu, J.; Huan, Y.; Yan, Y.; Huang, B.; Guo, Y.; Zheng, L.; Zou, Z. Self-aware distributed deep learning framework for heterogeneous IoT edge devices. *Future Gener. Comput. Syst.* **2021**, *125*, 908–920. [CrossRef]
26. Ji, T.; Luo, C.; Yu, L.; Wang, Q.; Chen, S.; Thapa, A.; Li, P. Energy-efficient computation offloading in mobile edge computing systems with uncertainties. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 5717–5729. [CrossRef]
27. Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.R.; Kaashoek, M.F.; Dabek, F.; Balakrishnan, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.* **2003**, *11*, 17–32. [CrossRef]
28. Panda, S.K.; Lin, M.; Zhou, T. Energy-efficient computation offloading with DVFS using deep reinforcement learning for time-critical IoT applications in edge computing. *IEEE Internet Things J.* **2022**, *10*, 6611–6621. [CrossRef]

29. Khan, M.A.; Yeh, L.; Zeitouni, K.; Borcea, C. MobiStore: A system for efficient mobile P2P data sharing. *Peer-to-Peer Netw. Appl.* **2017**, *10*, 910–924. [CrossRef]
30. Ye, Y.; Hu, R.Q.; Lu, G.; Shi, L. Enhance Latency-Constrained Computation in MEC Networks Using Uplink NOMA. *IEEE Trans. Commun.* **2020**, *68*, 2409–2425. [CrossRef]
31. Shinde, S.S.; Bozorgchenani, A.; Tarchi, D.; Ni, Q. On the design of federated learning in latency and energy constrained computation offloading operations in vehicular edge computing systems. *IEEE Trans. Veh. Technol.* **2021**, *71*, 2041–2057. [CrossRef]
32. Kiranyaz, S.; Pulkkinen, J.; Gabbouj, M. Multi-dimensional particle swarm optimization in dynamic environments. *Expert Syst. Appl.* **2011**, *38*, 2212–2223. [CrossRef]
33. Zhang, P.; Su, Y.; Li, B.; Liu, L.; Wang, C.; Zhang, W.; Tan, L. Deep Reinforcement Learning Based Computation Offloading in UAV-Assisted Edge Computing. *Drones* **2023**, *7*, 213. [CrossRef]