Delay Optimization Strategy for Service Cache and Task Offloading in Three-Tier Architecture Mobile Edge Computing System

LIMIAO LI¹ AND HENG ZHANG²

¹School of Computer Engineering and Applied Mathematics, Changsha University, Changsha 410003, China
²School of Computer Science and Engineering, Central South University, Changsha 410075, China

Corresponding author: Heng Zhang (405564123@qq.com)

This work was supported in part by the Natural Science Foundation of Hunan Province under Grant 2020JJ4647, in part by the outstanding youth project of the Hunan Education Department under Grant UQ19B056, in part by the general project of Hunan Education Department under Grant 19C0171, and in part by the general project of Changsha Technology Bureau under Grant KC1809015.

ABSTRACT As more and more compute-intensive and delay-sensitive applications are deployed on smart mobile devices, mobile edge computing is considered an effective way to solve the limited computing ability of smart mobile devices (SMDs). At present, latency has become the most critical indicator of the quality of service (QoS), and more and more studies focus on this aspect. Unlike previous work, our work fully takes into account the limited storage and computing ability of edge servers. To effectively reduce the delay of SMDs and improve QoS, we propose a Delay Control Strategy Joint Service Caching and Task Offloading (DCS-OCTO) in a three-tier mobile edge computing (MEC) system consist of multi-user, multi-edge server and remote cloud servers. Some of the key challenges include service heterogeneity, unknown system dynamics, spatial demand coupling, and decentralized coordination. In particular, a very compelling but rarely studied issue is the dynamic service caching in the three-tier MEC system. The DCS-OCTO strategy is proposed based on Lyapunov optimization and Gibbs sampling. It works online without requiring prior information and achieves provable near-optimal performance. Finally, simulation results show that the strategy effectively reduces the overall system delay while ensuring low energy consumption.

INDEX TERMS Mobile edge computing, Internet of Things, delay, service cache, task offloading.

I. INTRODUCTION Nowadays, with the rapid development of the internet of things (IoT) and wireless technology, more and more smart mobile devices (SMDs) are showing explosive growth [1], [2]. At the same time, SMDs often deploy applications that require supercomputing power, ultra-low latency, and persistent access rights, such as virtual reality and interactive online games. However, based on the consideration of the portability of SMD, the small size of SMD leads to faster energy consumption, weak computing power, and small storage space. This shortcoming of SMD severely hinders the deployment of a large number of applications [3]. Therefore, SMD attempts to overcome the shortcomings of itself by utilizing the supercomputing power and super storage capacity of cloud computing resources, that is, connecting to a remote cloud through a wireless network and offloading computing tasks to the remote cloud [4]. However, remote cloud computing resources are usually deployed in large data centers far from most users. Therefore, this will cause SMD to have a longer delay and higher energy consumption during the offloading process [5].

To solve the above multiple problems, the concept of mobile edge computing (MEC) has been proposed in recent years. MEC refers to the deployment of edge servers or computing nodes around the network [6]. Therefore, edge servers have a stronger computing capability than SMDs, and at the same time, edge servers are closer to users than remote cloud servers. Because of the characteristics of MEC, it can provide users with a short delay and low energy consumption services [1], [2], [7]. Generally, edge servers are deployed at access points (AP) and base stations (BS). Then, SMDs will offload the computing tasks or related data to the APs, and then the computing tasks after reaching the APs will be processed by the edge server deployed on the APs [8]. Currently, with the exponential growth of users’ quality of
service (QoS) requirements, the optimization of delay and energy consumption in a MEC system has become one of the most needed work in the MEC field [9], [10].

On the one hand, optimizing the service cache of edge server can effectively balance latency and energy consumption, and achieve the goal of improving users’ QoS [11], [12]. Generally, the service cache refers to an application program and its related data cached on the edge server to support the computation of the corresponding tasks. The effective service caching can greatly reduce the delay and energy consumption of users when users offload tasks [13]. However, many current works from the perspective of service caching assume that the edge servers have sufficient storage capacity and are capable of cache all services. This perspective ignores the storage capacity limit of the edge servers and does not meet the actual application scenario. Therefore, only a limited number of services can be cached on the edge servers. And because the storage space required by different services and the popularity of the tasks corresponding to the services are different, that is, the services have strong heterogeneity, it is necessary to specify which services are cached to maximize the performance of the MEC system.

On the other hand, the optimization of task offloading is also an indispensable part of balancing the delay and energy consumption of the MEC system [14], [15]. With the widespread popularity of SMDs, users hope that SMD can run desktop-level applications at any time and any place [10]. The current task offloading work often only considers the communication between the user and the edge server, that is, the users offload the tasks to the edge servers via the uplink, but does not consider that the computing power of the edge server is also limited. Therefore, it is necessary to establish a three-tier MEC system consisting of users, edge servers, and remote cloud servers, and establish a decision whether to offload a part of the computation tasks that the MEC servers cannot handle to the remote cloud servers.

Therefore, in this paper, to improve the overall performance of the MEC system, we established a three-layer MEC system consisting of users, APs, and remote cloud servers. Then, a joint optimization scheme for service caching and task offloading was developed to achieve the lowest latency of the system under the guarantee of energy consumption. The main contributions of this paper are shown below.

- Starting from the overall performance of the MEC system, we fully consider the limited storage space and computing capacity of the edge servers and established an SMDs-APs-remote servers three-layer MEC system to facilitate the offloading of computing tasks on APs. As far as we know, we take the lead in considering the comprehensive limitation of the storage space and computing capacities of the edge servers in an SMDs-APs-remote servers three-layer MEC system.

- In the SMDs-APs-remote servers three-layer MEC system, under considering the storage space and computing capabilities of the MEC servers, we study the problem of minimizing system latency under the guarantee of energy consumption constraint of joint service caching and task offloading. Moreover, this problem is formulated as a mixed-integer nonlinear optimization problem. Specifically, the service cache optimization is to determine which services the APs cache, and the task offload optimization is to determine how much the APs will offload their tasks to the remote cloud servers.

- Based on the Lyapunov optimization theory and Gibbs sampling, we developed a DCS-OCTO algorithm. This algorithm process random online service caching without prior knowledge. At the same time, compared with the optimal algorithm, the DCS-OCTO algorithm achieves that the average time delay is close to optimal while the energy consumption constraint is guaranteed in the three-layer MEC system.

- To verify the performance of the algorithm, we conducted a large number of simulation experiments to compare and analyze the DCS-OCTO algorithm with other algorithms. Considering the system delay and energy consumption from the perspective of APs storage space and energy consumption constraint, and the superiority of the algorithm is finally verified.

II. RELATED WORKS

At present, more and more research work focuses on the delay and energy consumption of MEC systems [16]–[19]. Many researchers mainly consider the optimization of system delay and energy consumption from two aspects of service caching and task offloading [20]–[22]. Dynamic service caching enables highly popular applications and their data among mobile users initially stored in the edge servers. In [23], Shuai Yu et al. proposed the best offloading scheme with a cache enhancement scheme (OOCs) for MEC scenarios. This scheme is suitable for single and multiple users, and the system execution delay is significantly reduced. Besides, compared with single-user MEC, multi-user MEC makes the system delay lower. The work [24] applies machine learning to MEC to promote efficient content caching. Among them, they mainly proposed a learning-based collaborative caching strategy (LECC) to reduce transmission costs while improving the quality of experience for mobile users. However, to be more in line with actual application scenarios, these efforts still need to fully consider the limitations of edge server storage space.

Besides, task offloading is a means to alleviate the limitations of mobile device resources, and computing tasks are shifted by SMDs to servers with stronger storage and computing capabilities. To improve the performance of mobile devices and reduce system latency and energy consumption, researchers have conducted in-depth research on computational offloading in MEC systems. But these efforts usually ignore the limited computing power of edge servers compared to remote cloud servers. In [25], Thai T. Vu et al. combined the task offloading and resource allocation optimization problems, and formulated the problem as a mixed-integer nonlinear programming. The problem was finally solved by the
proposed ROP and IBBA and finally realized the lowest total energy consumption of the users under the delay constraint guarantee. In [26], Ling Tang et al. studied the multi-user computing offload problem in an uncertain wireless environment and decided on whether or not the user offload task as a non-cooperative game based on the prospect theory (PT). And through the proposed distributed computing offloading algorithm, the game’s Nash equilibrium is achieved, which effectively helps mobile devices deal with computation-intensive and delay-sensitive tasks. Although in work [13], Jie Xu et al. considered the cache space limitation of the edge server and proposed an OREO algorithm in a dense cellular network to jointly optimize service cache and task offload, and finally achieved the lowest system latency under the guarantee of energy consumption. However, work [13] only considered the limitation of the storage space of the edge cloud, but failed to consider that the edge servers could only handle limited computing tasks within a certain time, that is, the limited computing capacity of the edge cloud.

Even in a small amount of work, the limited storage space and computing power of the edge server are fully considered, but this is not conducive to alleviating the computing pressure of mobile devices to a certain extent. In [27], Yixue Hao et al. studied the joint optimization problem of task caching and offloading on the edge cloud under the constraints of computing and storage resources. This is a difficult mixed integer programming problem. They designed an algorithm based on alternating iterations. The efficient algorithm, namely the task caching and offloading (TCO) algorithm, which solves this problem and ultimately reduces the energy consumption of the system. To fully alleviate the computing pressure of mobile devices, the current research should shift attention to the three-tier MEC system consisting of mobile devices, edge servers, and cloud servers [28]. Furthermore, we should allow the edge server to offload some incapable tasks to the cloud server again.

The rest of this paper is shown below. Section 3 describes the model of the system from architecture, service cache and task offload, delay, and energy consumption. Also, the problem of minimum delay subject to energy consumption is formulated in Section 3. Section 4 proposes the DCS-OCTO algorithm based on Lyapunov optimization and Gibbs sampling. This algorithm solves the target problem and its performance is fully analyzed. Section 5 fully validates the effectiveness of our proposed algorithm from various angles through simulation experiments. Finally, Section 6 concludes this paper.

### III. SYSTEM MODEL DESIGN

#### A. ARCHITECT MODEL

We consider a MEC three-tier architecture system with multiple users, multiple edge servers, and multiple remote cloud servers. The system consists of $N$ SMDs, $M$ APs integrating MEC servers and communication circuits, and several remote cloud servers. We use $\mathcal{N} = \{1, 2, \ldots, N\}, \forall n \in \mathcal{N}$ and $\mathcal{M} = \{1, 2, \ldots, M\}, \forall m \in \mathcal{M}$ to denote the set of SMDs and the set of APs, respectively. Besides, we define a service set $\mathcal{K} = \{1, 2, \ldots, K\}, \forall k \in \mathcal{K}$ whose total number of services is $K$. For more intuitive expression, Figure 1 shows the system architecture. As can be seen from Figure 1, because the deployment of APs is often dense, each SMD can access a group of APs within the broadcast range. For example, the APs accessible by SMD 2 are AP 1 and AP 4. Specially, we define a set $\mathcal{U}_n \in \mathcal{M}$ to represent a group of APs that SMD $n$ can access. From this, we know an SMD can only offload computing tasks to APs that cache the corresponding service and are within the access range of the SMD. However, when all APs in the accessible range of the SMD does not cache the services corresponding to the tasks, the SMD needs to offload the tasks directly to the remote cloud server, such as SMD 2. Also, if there are offloaded tasks from SMDs that an AP cannot handle, the AP can offload these tasks to the remote cloud servers, such as AP 1 and AP 3. For convenience, the definitions of the main symbols involved in this paper are shown in Table 1.

#### TABLE 1. Definition of key mathematical notations.

| Symbol | Definition |
|--------|------------|
| $\mathcal{N}$ | The set of SMDs |
| $\mathcal{M}$ | The set of APs |
| $\mathcal{K}$ | The set of services |
| $\mathcal{U}_m$ | The set of APs accessible by SMDs $n$ |
| $C_m$ | The storage space of AP $m$ |
| $f_m$ | The maximum frequency of AP $m$ when processing tasks |
| $c_k$ | The storage space required for service $k$ |
| $\omega_k$ | The workload average followed by service $k$ |
| $\lambda_k^{n, t}$ | The computation demand of service $k$ on SMD $n$ at time slot $t$ |
| $\lambda_k^t$ | The computation demand vector of service $k$ at time slot $t$ |
| $b^s_{m,k}$ | The cache decision of service $k$ on AP $m$ at time slot $t$ |
| $b^c_{m,k}$ | The service cache decision of AP $m$ at time slot $t$ |
| $b^t_{m,k}$ | The service cache decision of all APs at time slot $t$ |
| $\gamma_{m,k}$ | The set of APs that cache service $k$ at time slot $t$ |
| $\gamma_{m,k}$ | The computation amount of service $k$ on AP $m$ at time slot $t$ |
| $d_m^r$ | The task offload decision of AP $m$ at time slot $t$ |
| $d^t$ | The task offload decision of all APs at time slot $t$ |
| $\varphi_m$ | The static energy consumption of AP $m$ |
| $\eta_m$ | The unit energy consumption required to process computing tasks of AP $m$ at maximum processing speed |
| $E_m^c(b^s, d^t)$ | The average computed energy consumption of AP $m$ in time slot $t$ |
| $r_{m,k}$ | The actual computation amount corresponding to AP $m$ in time slot $t$ |
| $r_{m,k}$ | The actual total workload of AP $m$ at time slot $t$ |
| $s$ | The random variable of service time |
| $E[s]$ | The energy consumption at the first moment |
| $E[s]^r$ | The energy consumption at the second moment |
| $\tau_{m,k}$ | The delay of AP $m$ processing tasks in time slot $t$ |
| $\tau_{m,k}$ | The delay for tasks to be transferred to the remote cloud server in time slot $t$ |
| $T_{m,k}(b^s, d^t)$ | The total delay caused by AP $m$ in time slot $t$ |

Specifically, the MEC server in the AP is responsible for service caching and task computation, and the communication unit in the AP is responsible for communication between the MEC server and SMDs, that is, receiving tasks that SMDs offload through the uplink and receiving the results returned by the server and returns the results to the SMDs via the downlink. The MEC server integrated into the AP $\forall m \in \mathcal{M}$
usually stores services and data related to computing tasks, and the storage space is $C_m$. Services related to computing tasks mainly refer to applications or algorithms cached on the AP and requested by the SMD, such as social games, navigation, and video streaming. Data related to computing tasks refers primarily to the libraries and databases needed to run a particular application or algorithm. It should be emphasized that in the following, we will not separately describe the specific functions of the components in the AP, such as caching, task computing, and communication, but will attribute these functions to the AP. For example, when AP $m$ processes tasks offloaded from the SMD, the maximum clock frequency is $f_m$ (cycles/second). And the storage space and computing power of AP $m$ are $C_m$ and $B_m$, respectively. As for the computing resources allocated to each task by AP $m$ is $g_m$.

As for each specific service type in the system, the storage space of the service $k$ is $c_k$, and the workload of the corresponding task follows an exponential distribution with an average value $\omega_k$. It can be seen that the storage space occupied by the service and the required central processing unit (CPU) cycles are heterogeneous and irrelevant. As for the total system operation time, it is divided into $s$ time slots, which are represented by a set $T = \{1, 2, \ldots, s\}, \forall t \in T$, where the length of time slot $t$ is $T$. In time slot $t$, the computing demand strength of service $k$ for SMD $n$ is defined as $\lambda_{t,k,n}$, that is, the computing task on SMD $n$ follows the Poisson process with the rate of $\lambda_{t,k,n}$. Therefore, the computing demand vector of service $s$ for $N$ SMDs in time slot $t$ can be expressed as $\lambda_{t,k} = (\lambda_{t,k,1}, \lambda_{t,k,2}, \ldots, \lambda_{t,k,N})$. At the beginning of each time slot $t$, the AP decides its own service cache decision and the decision to offload the task from the AP to the remote cloud server. It should be emphasized that our description below only needs to focus on a single time slot. Next, the service cache and task offload models will be described in detail.
B. SERVICE CACHE AND TASK OFFLOAD MODEL

In this section, we will describe the service cache model of the AP in the MEC three-tier architecture system with multiple users, multiple edge servers, and multiple remote cloud servers, and the task offload model of the AP offloading tasks to the remote cloud server.

1) SERVICE CACHE MODEL

The service cache model establishes the cache decision of all APs in the system for each service type. We assume that AP \( m \) caches service \( k \) and AP \( m \) is in the accessible range of SMD \( n \), namely \( m \in \mathcal{U}_n \). Then when there is a task corresponding to service \( k \) on SMD \( n \) that needs to be processed, SMD \( n \) only needs to offload the input data of the task to AP \( m \) where service \( k \) is cached, no need to offload the entire task (such as program or algorithm) to the AP \( m \) via the uplink. After AP \( m \) finishes computing the task, the computation result is then returned to SMD \( n \) through the downlink. Because of this, the service cache can effectively reduce the delay and energy consumption of the SMDs in the system during the uplink offloading process, and greatly improve the user service quality. However, the edge server is different from the remote cloud server in that its storage capacity is much lower than the remote cloud server. As mentioned earlier, the storage space of AP \( m \) is \( C_m \). To effectively utilize the limited storage space of edge servers, all APs must make optimal service caching decisions to determine which services to cache and which not. Specifically, we define a binary decision variable \( b_{m,k}^t \in \{0, 1\} \) to indicate whether AP \( m \) caches service \( k \) in time slot \( t \). When \( b_{m,k}^t \) is set to 1, it means that AP \( m \) caches service \( k \) in time slot \( t \). When \( b_{m,k}^t \) is set to 0, it means that AP \( m \) does not cache service \( k \) in time slot \( t \). In fact, the service set cached by AP \( m \) is a subset of \( K \). We defined the service cache decision of AP \( m \) in time slot \( t \) as a set \( B_m^t = \{ b_{m,k}^t \} \), and the service cache decision of all APs in time slot \( t \) in the system is represented by \( B^t = \{ b_{m,k}^t \} \). Therefore, the storage space capacity limit that the total AP \( m \) cache sum needs to meet in time slot \( t \) is then obtained. And the limitation is shown as follows

\[
\sum_{k=1}^{K} b_{m,k}^t c_k \leq C_m, \quad \forall t, \forall m. \tag{1}
\]

In more detail, we define \( \mathcal{V}_{n,k}^t \subseteq \mathcal{U}_n \) to represent the set of APs that accessible by SMD \( n \) and cached service \( k \) in time slot \( t \). Therefore, when SMD \( n \) needs to process the task corresponding to service \( k \), it will offload the task to APs in the set \( \mathcal{V}_{n,k}^t \) during time slot \( t \). It should be noted that we did not offload all the requests corresponding to a service to a certain AP, because this would cause the AP to be severely overloaded, which would affect the computing efficiency, the overall performance of the system, and ultimately reduce the user service quality. Even an AP that is severely overloaded can offload some of the requests to a remote cloud server, which will have a huge impact on system performance. Therefore, in this paper, we consider to uniformly allocate the requested amount \( \lambda_{n,k}^t \) on SMD \( n \) to the APs that SMD \( n \) can access and cached service \( k \) in time slot \( t \), that is, \( \mathcal{V}_{n,k}^t \). Furthermore, the task computation amount corresponding to service \( k \) on AP \( m \) in the set \( \mathcal{V}_{n,k}^t \) can be easily obtained as

\[
r_{m,k}^t = b_{m,k}^t \sum_{n \in \mathcal{N}} \left\lfloor \frac{\lambda_{n,k}^t}{|\mathcal{V}_{n,k}^t|} \right\rfloor \tag{2}
\]

Next, we describe the task offload model.

2) TASK OFFLOAD MODEL

The task offload model establishes the decision of the AP whether to transfer the number of tasks that have not been processed to the remote cloud server. In fact, in addition to the storage space is much lower than the remote cloud server, the computing power of the AP is also much weaker than the remote cloud server. Therefore, it is necessary to offload some tasks to the remote cloud server under certain circumstances. For example, if there is no AP cache service \( k \) in the AP set \( \mathcal{U}_n \) accessible by SMD \( n \), or the AP currently does not provide enough computing power to handle the number of tasks corresponding to service \( k \), then the APs in the set \( \mathcal{U}_n \) will offload the tasks to a remote cloud server and use the strong computing power of the remote cloud server to process tasks. We define the offloading decision variable of AP \( m \) at time slot \( t \) as \( d_m^t \in \{0, 1\} \), which is continuous and is used to represent the proportion of computing tasks handled by AP \( m \) itself in time slot \( t \). When \( d_m^t = 0 \), it means that AP \( m \) does not handle any computing tasks in time slot \( t \), but all tasks were offloaded to the remote cloud server. When \( d_m^t = 1 \), it means that AP \( m \) does not offload any tasks to the remote cloud server, and all tasks offloaded from SMDs are handled by AP \( m \) itself. As for the offloading decision of all APs in time slot \( t \), it is represented by a set \( d^t = (d_1^t, d_2^t, \ldots, d_M^t) \). According to the offloading decision of APs, we can easily get the number of tasks processed by each AP. In time slot \( t \), the number of tasks processed by AP \( m \) itself is \( d_m^t \sum_{k \in K} l_{m,k}^t \). Besides, the total computing power required by AP \( m \) for processing tasks in time slot \( t \) needs to satisfy the inequality corresponding to its computing power limitation, that is,

\[
d_m^t \sum_{k \in K} l_{m,k}^t \cdot g_k \leq G_m, \quad \forall t, \forall m. \tag{3}
\]

It should be mentioned that the process of offloading tasks from the AP to the remote cloud server depends not only on different task requirements but also on the time slot when the tasks reach the AP. Moreover, based on the non-prior knowledge, and based on the proportion of offloaded tasks, it is feasible to plan the task offloading of each time slot with reasonable high granularity.

C. ENERGY CONSUMPTION AND DELAY MODEL

Different service cache strategies and different task offload strategies of APs in the system will bring different levels of energy consumption and delay overhead to the SMDs in the
system. In this section, we introduce the energy consumption model and latency model in detail and quantify the impact of service caching strategies and task offloading strategies on energy consumption and latency.

1) ENERGY CONSUMPTION MODEL

First, we introduce the energy consumption model of the AP. The energy consumption of AP mainly comes from three phases. The three phases are the computation of the tasks offloaded from the SMDs by the APs, the transmission of the computation results by APs via the downlink, and the offload of the tasks that the APs cannot handle to the remote cloud server. For the phase that the AP process the tasks offloaded from the SMD, although the AP often dynamically adjusts its CPU execution speed according to different task loads, we assume that the AP processes tasks at the maximum CPU execution speed in this paper and the CPU computation speed of the AP in the idle state is the minimum CPU. For the phase where the AP transmits the result of the task execution to the SMD via the downlink, because the size of the returned data result is often much smaller than the amount of the task transmitted by the uplink, and the distance between the SMD and the AP compared with the distance between SMD and the remote cloud server is much smaller, so the delay and energy consumption at this stage is often not considered in many previous works. We also ignore the delay and energy consumption of the downlink [3], [29]. As for the last phase is because in addition to AP can handle tasks offloaded from the SMD, there are also services that APs do not cache, or APs do not have enough computing power. This last phase is based on high-speed wired networks [30], so the energy consumption in this phase can be ignored. Based on the previous model, the energy consumption of the AP in time slot $t$ can be easily obtained. The following equation shows the energy consumption of AP $m$ in time slot $t$ under service cache strategy $b^t$ and task offload strategy $d^t$.

$$E_m^t(b^t, d^t) = e_m d_m^t \sum_{k \in \mathcal{K}} \omega_k b^t_{m,k} r^t_{m,k} + \varphi_m. \quad (4)$$

Among them, $e_m$ is the unit cycle energy consumption when AP $m$ processing tasks under the maximum CPU execution speed $f_m$, and $d_m^t \sum_{k \in \mathcal{K}} \omega_k b^t_{m,k} r^t_{m,k}$ is the total number of CPU cycles required by AP $m$ to process computation tasks in time slot $t$, so $e_m d_m^t \sum_{k \in \mathcal{K}} \omega_k b^t_{m,k} r^t_{m,k}$ is the total energy consumption of AP $m$ process computation tasks in time slot $t$. Also, as long as the AP is running, a small amount of static energy consumption that is not related to the task load will be generated in each time slot. We define it as $\varphi_m$, which is the static energy consumption of AP $m$ in time slot $t$. In addition to the operation of computing tasks, AP also has operations that are not related to computing tasks. And we define it as $\tilde{E}_m^t$, which represents the operation of AP $m$ that is not related to computing tasks in time slot $t$. In particular, such energy consumption varies over time and can only be observed at the end of a time slot.

It is important to note that the AP must not only meet its energy consumption constraints in a single time slot but also meet the overall energy consumption constraints in the total time. The energy consumption of AP $m$ in time slot $t$ cannot exceed its maximum energy consumption value $E_m^{max}$, that is,

$$E_m^t(b^t, d^t) \leq E_m^{max}. \quad (5)$$

The total energy consumption constraint of AP $m$ cannot exceed $E$, that is,

$$\lim_{s \to \infty} \frac{1}{s} \sum_{t \in T} \sum_{m \in \mathcal{M}} (E_m^t(b^t, d^t) + \tilde{E}_m^t) \leq E \quad (6)$$

In this paper, we construct a virtual energy shortage queue for APs in the system, and the energy shortage queue is updated over time. Specifically, we define $Q(t)$ as the backlog of energy shortage queue in time slot $t$, which is the difference between the energy consumption and energy constraint $E$ in current time slot $t$. Based on the above existing analysis, assuming $Q(0) = 0$, the dynamic evolution of APs energy shortage queue can be expressed as

$$Q_m(t+1) = \max\left\{ \sum_{m \in \mathcal{M}} (E_m^t(b^t, d^t) + \tilde{E}_m^t) - E, 0 \right\} + Q_m(t) \quad (7)$$

2) DELAY MODEL

In addition to quantifying the energy consumption in network overhead, SMD needs to establish a delay model and quantify and reduce the delay, which is conducive to improving the overall network performance and enhancing the user experience. We build latency costs primarily by calculating user dissatisfaction caused by latency and lost revenue. The delay caused by the AP are mainly the results of task computation, task offload, and the results returned. As explained in the previous section, the delay of the results returned by the downlink is negligible, so this paper does not consider the delay of this process. As for the delay caused by the SMDs, it is mainly because the task is offloaded through the uplink. This article does not consider the delay when the SMD offloads the task to the AP, because the distance between the SMD and the AP is much closer than the distance between the SMD and the remote cloud server. Therefore, only the delay in the process of offloading computing tasks to the remote cloud server by SMD is considered. Only when the SMD offloads the task amount to the AP that has the corresponding service cached in its area, but the remaining part of the task amount remains, the SMD will offload the remaining task to the remote cloud server.

First, regarding the task computation of the AP, the AP will process a series of computation tasks offloaded from the SMDs. We model the service process of the AP as an M/G/1 queue (M stands for the exponential distribution, and the exponential distribution is memorylessness, that is Markov; G is the distribution of general service time; 1 is the single service desk, waiting system). And the average
time spent in the queue (the sum of service time and waiting time) is analyzed to calculate the average delay of the task computation process. Because it was previously defined that the task arrival corresponding to different services follows the Poisson distribution, the entire process of task arrival in the system also follows the Poisson distribution. According to the service cache decision of the AP, it is easy to obtain the task load on each AP. As introduced earlier, $r_{m,k}^t$ represents the task computation amount corresponding to service $k$ on AP $m$ in time slot $t$, then $r_{m,k}^t = \sum_{k \in K} r_{m,k}^t$ is the total task computation amount of AP $m$ during time slot $t$. Next, based on the task offloading decision, the actual task amount corresponding to service $k$ on AP $m$ in time slot $t$ is $r_{m,k}^t \sim \frac{d_i^t}{r_m^t} \cdot r_m^t$. The actual total task load on AP $m$ in time slot $t$ is $r_m^t = \sum_{k \in K} r_{m,k}^t$. Therefore, the amount of tasks that AP $m$ offloads to the remote cloud server is $r_m^t - r_{m,k}^t$. To meet the various characteristics of service type in real application scenarios, the total time distribution of various services is derived from an exponential random sampling. For example, the total time distribution of service $k$ is derived from a random sampling of an exponential distribution with an average of $\omega_k$. Therefore, the exponential distribution probability corresponding to service $k$ in AP $m$ during time slot $t$ is $\frac{r_{m,k}^t}{r_m^t}$. Then the total energy consumption of AP $m$ at the first moment can be obtained, which is

$$E[s] = \sum_{k \in K} \omega_k \frac{r_{m,k}^t}{r_m^t}.$$  

(8)

As for the total energy consumption for the second moment, it is

$$E[s^2] = \sum_{k \in K} 2\omega_k^2 \frac{r_{m,k}^t}{r_m^t}.$$  

(9)

At the same time, according to the P-K formula in the M / G / 1 queueing system [31], we can get the average time (sum of service time and wait time for the AP to process computation tasks in each time slot). The following equation (10) represents the delay by AP $m$ to process the unit task amount under the service cache strategy $b^t$ and task offload strategy $d^t$.

$$\tau_m^t(b^t, d^t) = \frac{1}{v_m} E[s] + \frac{r_m^t \cdot E[s^2]}{2(v_m - r_m^t \cdot E[s])},$$  

(10)

which $v_m$ is the service rate of AP $m$.

Secondly, concerning the task offloading of the AP, when the AP cannot support excessive computing load, the AP will transfer the excess computation task to a remote cloud server with strong computing capabilities. Therefore, we define the transmission delay of the unit task amount of the AP and the SMD to offload the computing task to the remote cloud server in the time slot $t$ as $d^t$. Then, the total transmission delay of offloading the computation task to the remote cloud server of AP $m$ during time slot $t$ is $d^t(r_m^t - \tau_m^t)$. In the end, we can get that the total delay composed of task computation delay and offload delay for AP $m$ in time slot $t$ is

$$T_m^t(b^t, d^t) = \tau_m^t + d^t(r_m^t - \tau_m^t)$$

$$= \frac{\tau_m^t}{f_m} \cdot \sum_{m \in M} \omega_k \frac{r_{m,k}^t}{r_m^t} + \frac{\tau_m^t \cdot r_m^t}{f_m} \sum_{k \in K} \omega_k \frac{r_{m,k}^t}{r_m^t}$$

$$+ d^t r_m^t - d^t \tau_m^t$$

$$= d^t r_m^t + \sum_{k \in K} (\omega_k - d^t) \frac{r_{m,k}^t}{r_m^t} + \frac{\sum_{k \in K} \omega_k r_{m,k}^t}{1 - \sum_{k \in K} \omega_k}$$

(11)

(16)

In time slot $t$, the delay of all SMDs in the system due to the transmission of the remaining tasks to the remote cloud server is

$$T_{SMDs}^t = d^t \left( \sum_{m \in M} \sum_{k \in K} \lambda_k^t - \sum_{n \in N} r_n^t \right).$$  

(12)

Therefore, the total delay caused by APs and SMDs in the system during time slot $t$ is expressed as follows

$$T_2^t(b^t, d^t) = T_m^t(b^t, d^t) + T_{SMDs}^t.$$  

(13)

It is important to note that the AP should satisfy the delay constraint in each time slot. The delay of AP $m$ in time slot $t$ cannot exceed $T_{m,max}$, that is,

$$T_m^t(b^t, d^t) \leq T_{m,\text{max}}.$$  

(14)

D. OPTIMIZATION PROBLEM FORMULATION

Based on the introduction of the above model, we know that the system delay is composed of APs and SMDs. Now, we express the average delay of the system under the total time slot number $s$ with the following equation.

$$T = \lim_{s \to \infty} \frac{1}{s} \cdot E[\sum_{t=0}^{s-1} T_1^t(b^t, d^t)].$$  

(15)

To minimize the average delay of the total system time under the premise of ensuring energy consumption, we jointly optimize the APs service cache strategy and APs task offload strategy in the system. And we can easily get the expression with the minimum average delay in the total time of the system and it is shown below,

$$P1 : \min_{b^t, d^t} T$$

s.t. (1), (3), (5), (6), (14).  

(16)
Markov Chain Monte Carlo (MCMC), we design an online algorithm that does not need to foresee future information and combines service cache decision and task offload decision. In the next section, we will give a detailed solution to the problem of $P1$.

IV. ONLINE SERVICE CACHE AND TASK OFFLOADING STRATEGY

In this section, we first propose an online algorithm based on Lyapunov and MCMC [32], that is the DCS-OCTO algorithm. By the Lyapunov optimization technology, the original problem $P1$ can be transformed into a problem that is easier to solve. Then, we make full use of the advantages of MCMC in sample collection and solving the optimization problem to solve the transformed problem. This algorithm transforms the long-term optimization problem into the delay optimization problem of each time slot, which determines which services are cached in APs and how many tasks are reserved for processing on APs. Therefore, we only need to focus on each time slot.

A. PROBLEM TRANSFORMATION BASED ON LYAPUNOV

First, we define the Lyapunov function $L(Q(t))$, which is the square of the energy shortage queue. The specific Lyapunov function is as follows

$$L(Q(t)) = \frac{1}{2} Q^2(t).$$ (17)

The corresponding value of $L(Q(t))$ indicates the congestion level of the energy shortage queue. A smaller $L(Q(t))$ value represents a smaller backlog of the virtual energy shortage queue, which further means that the queue is more stable. Conversely, a larger $L(Q(t))$ value means a larger queue backlog and stability weaker. To ensure that the energy shortage queue has a small backlog and strong stability, we further define Lyapunov drift $\Delta L(Q(t))$ to continue to promote to a lower value. $\Delta L(Q(t))$ is shown below

$$\Delta L(Q(t)) = \mathbb{E}[L(Q(t + 1)) - L(Q(t))]Q(t)].$$ (18)

Next, $\Delta L(Q(t))$ was expanded and transformed. Theorem 1 gives the upper bound of $\Delta L(Q(t))$ in time slot $t$.

**Theorem 1:** Let $\hat{E}^i(b^i, d^i) = \sum_{m \in M}(E_{m}^i(b^i, d^i) + \hat{E}_{m}^i)$, the upper bound of $\Delta L(Q(t))$ is

$$\Delta L(Q(t)) \leq B + Q(t)\mathbb{E}[\hat{E}^i(b^i, d^i) - E(Q(t)],$$ (19)

which $B = \frac{1}{2}(\sum_{m \in M} E_{m}^{max} - Q^2)$

**Proof:** The proof is provided in Appendix A. $\square$

The goal of the online algorithm is to minimize the overall system delay. Therefore, under the Lyapunov framework, we integrate the minimum delay objective function into the Lyapunov drift function $\Delta L(Q(t))$, thereby obtaining the drift plus delay expression $\Delta V L(Q(t))$ and its corresponding upper limit (20), i.e.,

$$\Delta V L(Q(t)) = \Delta L(Q(t)) + V \cdot \mathbb{E}[T^i(b^i, d^i)|Q(t)]$$

$$B + Q(t)\mathbb{E}[T^i(b^i, d^i) - E(Q(t)] + V \cdot \mathbb{E}[T^i(b^i, d^i)|Q(t)]$$ (20)

$V$ in inequality (20) is a non-negative weight used to control the proportion of delay in $\Delta V L(Q(t))$. As $V$ increases, the proportion of delay in $\Delta V L(Q(t))$ becomes larger; otherwise, the proportion of delay in $\Delta V L(Q(t))$ becomes smaller. It should note that although the smaller $V$ is, the shorter the delay will be, but the longer the energy shortage queue $Q(t)$ will be. However, once the energy shortage queue is too long, the basic energy of the system may not be guaranteed. Therefore, reasonable adjustment of the $V$ value and following the violation of the energy consumption constraint will lead to less available energy to maintain the balance between the delay and the energy shortage queue, and finally, achieve the goal of minimizing the delay of the system under energy guarantee.

To achieve the purpose of minimizing the delay under the energy guarantee, we only need to minimize the drift plus the delay. In other words, as long as the upper limit of the drift plus delay is minimum, the service cache strategy and task offload strategy of the APs in each time slot can be determined, and the purpose of minimizing the delay of the system under the energy guarantee is finally achieved. Besides, because we only focus on queue changes for a single time slot, we delete the right-hand expectation of inequality (20). Also, because $B$ is a constant, and finally we convert $P1$ to $P2$, as shown below

$$P2 : \min (Q(t) \cdot (\hat{E}^i(b^i, d^i) - E)

\hat{b}^i, d^i

\text{s.t.} (1), (3), (5), (6), (14).$$ (21)

The optimal solution of $P2$ can be obtained through the DCS-OCTO algorithm to get the service cache strategy and task offload strategy of the APs during each time slot in the system. The DCS-OCTO algorithm will be given in detail in the next section.

B. THE DCS-OCTO ALGORITHM

To solve $P2$, this section formulates a distributed optimization scheme for $P2$ and finally proposes the DCS-OCTO algorithm. First, $P2$ as a joint optimization problem, the optimal service cache decision and task offload decision of APs in the system can be determined in each time slot by solving $P2$. Through the introduction of the above model part, we know that the service cache decision and task offload decision are binary and continuous, respectively. Therefore, $P2$ is a mixed-integer nonlinear programming (MINLP) problem. To solve $P2$, we propose a distributed optimized DCS-OCTO algorithm based on a special case in MCMC, namely Gibbs Sampler [33]. The algorithm determines the service cache decision and task offload decision of each slot at the beginning of each slot in an iterative manner. And the average time complexity of the DCS-OCTO algorithm is $O(nlog_2 n)$. Next, the working steps of the DCS-OCTO algorithm are described
in detail, and algorithm 1 describes these steps in detail using pseudocode.

**Algorithm 1** The Proposed DCS-OCTO Algorithm

**Input:** $Q(t)$, $b_k$, $c_k$, $C_m$, $G_m$, $E_{max}$, $E$, $s$; service cache decision $b^t_m \leftarrow 0$; offloading decision $d^t_m \leftarrow 0$; object value $\delta \leftarrow \infty$ 

**Output:** $Q(t+1)$, $b^t_m$, $d^t_m$

1: Randomly pick a AP $m \in M$ and select service cache decision $b^t_m \in \Psi$;
2: if $b^t_m \in \Psi$ is feasible then
3: $b^t_m' \leftarrow \{b^t_{m-1}, b^t_m\}$;
4: Observe computing demand $r^t_{m,k}, \forall m, \forall k$;
5: Obtain $d^t_m', \forall k$ by minimizing $P2 : \min_V V \cdot T^t(b^t_m', d^t_m) + Q(t) \cdot E^t(b^t_m', d^t_m)$
6: $\varepsilon \leftarrow \frac{1}{1 + e^{t_V}}$
7: With probability $\varepsilon$, BS $m$ sets $b^t_m' \leftarrow b^t_m', d^t_m' \leftarrow d^t_m'$, $\delta \leftarrow \delta'$;
8: With probability $(1 - \varepsilon)$, BS $m$ keeps at $b^t_m'$ unchanged;
9: Broadcast $b^t_m'$ to its neighboring BSs;
10: end if
11: Return at $b^t_m', d^t_m$ if the stopping criterion is satisfied, otherwise, go to Line 2;

In general, in each iteration process, an AP is randomly selected, such as AP $m$, and the service cache decision $b^t_m$ of AP $m$ during time slot $t$ is modified to $b^t_m'$. With the solution of equation (21), task offloading decision $d^t_m$ can be obtained.

$\text{Observe computing demand } r^t_{m,k}, \forall m, \forall k; \text{ Obtain } d^t_m', \forall k \text{ by minimizing } P2 : \min_V V \cdot T^t(b^t_m', d^t_m) + Q(t) \cdot E^t(b^t_m', d^t_m)$

$\varepsilon \leftarrow \frac{1}{1 + e^{t_V}}$

With probability $\varepsilon$, BS $m$ sets $b^t_m' \leftarrow b^t_m', d^t_m' \leftarrow d^t_m'$, $\delta \leftarrow \delta'$;

With probability $(1 - \varepsilon)$, BS $m$ keeps at $b^t_m'$ unchanged;

Broadcast $b^t_m'$ to its neighboring BSs;

end if

C. ALGORITHM PERFORMANCE ANALYSIS

In this section, we analyze the performance of the DCS-OCTO algorithm. First, Theorem 3 is given as shown below.

**Theorem 3:** The delay at the average time of the system generated by the DCS-OCTO algorithm satisfies the following inequality.

$$\lim_{s \to \infty} \frac{1}{s} \sum_{t=0}^{s-1} \mathbb{E}[T^t(b^t_m', d^t_m)] < T^\text{opt} + \frac{B}{V}. \quad (22)$$

While the energy consumption of APs in the average time satisfies the following inequality, namely

$$\lim_{s \to \infty} \frac{1}{s} \sum_{t=0}^{s-1} \mathbb{E}[E^t(b^t_m', d^t_m)] < \frac{B}{\varepsilon} + \frac{V}{\varepsilon(T^\text{max} - T^\text{opt} + E).} \quad (23)$$

Among them, the minimum value of $P2$, that is, the optimal system delay is

$$\lim_{s \to \infty} \frac{1}{s} \sum_{t=0}^{s-1} \sum_{m=1}^{M} \mathbb{E}[T^t_m(b^{opt,t}_m, d^{opt,t}_m)], \quad (24)$$

and the longest system delay is $T^\text{max}$. As for $\varepsilon, \varepsilon > 0$ is the long-term energy residual constant, which is obtained by a fixed control strategy.

Proof: The proof is provided in Appendix C. \hfill \ensuremath{\Box}

The above theorem proves that the delay energy tradeoff is $[O(1/V), O(V)]$. By making $V \to \infty$, the DCS-OCTO algorithm asymptotically achieves the optimal performance of offline problem $P1$. However, the optimal performance of $P1$ comes at the cost of higher energy consumption, because a larger energy shortage queue is needed to stabilize the system, which delays convergence. This also means that the energy consumption at the average time increases linearly with $V$. 

VOLUME 8, 2020
V. SIMULATION

In this section, we use Matlab software to simulate and comprehensively evaluate the performance of the DCS-OCTO algorithm based on the simulation results. Finally, it is verified that the DCS-OCTO algorithm can effectively reduce the overall system delay while ensuring low energy consumption, and greatly improve QoS.

A. PARAMETERS SETTING IN SIMULATION

We simulated a rectangular area with a length and width of 400 m. There are 16 SMDs in the entire rectangular area. To provide services to the SMDs in the area, we deployed 9 APs with a radius of 145 m on the grid network in the area. During time slot $t$, the actual service demand of service $k$ on SMD $n$ is a Poisson process with an arrival rate of $\lambda_{k,n}$, and it follows the uniform distribution of $\lambda_{k,n} \in [0, 10]$. Specifically, we define the total number of service types $K = 8$, the storage space $c_k = [10, 120]$ GB required by service $k$, and the CPU clock range $g_k = [0.1, 0.6]$ GHz required by any service $k$ corresponding to a single task. Regarding the related parameter settings of the AP, the communication distance $L = 140$ is between APs. When specific APs are involved, we take AP $m$ as an example, the service rate is $v_m = 12$ GHz, and storage space is $C_m = 240$ GB. Also, the CPU clock is $G_m = 240$ GHz, and the unit energy consumption is $e_m = 1.2$ kWh. As for non-task processing energy consumption, it is $E_{\mu} = [0, 2]$ kWh. At last, the smoothing factor and the unit task volume transmission delay of tasks offloaded to the remote cloud server are $\eta = 10^{-2}$ and $a' = [2, 5]$ sec, respectively.

In the following content, we compare the DCS-OCTO algorithm with two benchmarks to evaluate the delay and energy consumption over time, in which the two benchmarks are Independent Service Cache Solution (Independent Strategy) and Delay-First Service Cache Solution (Delay-first Strategy). At the same time, we also start from the AP storage capacity constraints and compare the delay and energy consumption of different algorithms with a change of AP storage capacity at the average time. Next, we also evaluated the delay and energy consumption of the DCS-OCTO algorithm under the average time of different AP energy constraints. Finally, we show the variation of the convergence process of the DCS-OCTO algorithm with the smoothing coefficient. For the sake of understanding, we will explain the two benchmarks separately. Under Independent Strategy, APs only cache the service types with the highest demand in their service area, and APs do not communicate with each other and only work independently; while under Delay-first Strategy, to reduce system delay, all APs make use of centralized service caching decisions without considering long-term energy constraints [3], [35], [36].

B. ALGORITHM PERFORMANCE EVALUATION

Figure 2 and Figure 3 show the average time delay and energy consumption versus time slot $t$, respectively. It can be seen from Figure 2 and Figure 3 that the DCS-OCTO algorithm can achieve the optimal system delay under the condition of guaranteed energy consumption. Although the Delay-first Strategy can make the system reach the lowest latency, it comes at the expense of more energy consumption. As for the Independent Strategy, the delay and energy consumption are quite high because the solution does not consider the dependent cooperation relationship between APs, and only makes final decisions based on the predicted demand made by the APs alone.

Figure 4 and Figure 5 are the results obtained after 200-time slots (i.e., $s = 200$) simulation, which reflects the impact on the system delay and energy consumption under different storage capacities of APs (i.e., $C_m$), respectively. Importantly, we also compared the impact of different schemes. Specifically, it can be seen from Figure 4 that under the three schemes, the system delay decreases with the increase of storage space, and the system delay eventually stabilizes. This is because the larger the storage space of the APs, the more services can be cached, and then more computation tasks can be processed faster by using stronger computing power than SMDs. However, because the number of service types is limited, the delay eventually stabilizes. Even if the cache space grows, it will not cache other services. It is worth noting that the average time delay of the DCS-OCTO algorithm when the storage capacity of the APs is small is the
same as the Delay-first Strategy. This is because most of the tasks are offloaded to the remote cloud server for processing, while the APs themselves process very few tasks. Besides, the energy shortage queue $Q(t)$ takes 0 for a long time. This is the objective equation in $P^2$ becoming a latency-optimal service caching scheme. As for the DCS-OCTO algorithm, the average time delay is finally higher than the delay of Delay-first Strategy lies in that the system deviates from the delay after meeting the energy consumption constraint.

Then, as shown in Figure 5, except for the DCS-OCTO algorithm, the other two strategies far exceed the long-term energy consumption limit, and as the storage space grows, the energy consumption first rises and then stabilizes. The main reason is that more services will be cached after the storage space is increased, which will cause APs to process more tasks and eventually increase energy consumption. The reason why energy consumption stabilizes again is similar to the reason for the delay stabilization in Figure 4. It is also because APs no longer have more services to cache, and thus will not cause fluctuations in energy consumption again. It should be emphasized that, because the DCS-OCTO algorithm was designed to meet the energy consumption limit at the beginning, the energy consumption generated by the DCS-OCTO algorithm almost coincides with the long-term energy consumption limit.

Figure 6 and Figure 7 show the process of the delay and energy consumption of the system with the increase of the energy consumption limitation $E$ when the DCS-OCTO algorithm converges, respectively. Obviously, as the energy consumption limitation $E$ increases, that is, the energy consumption constraint becomes lower, then the DCS-OCTO algorithm will converge to a lower system delay with a higher energy consumption guarantee. In other words, as the energy consumption restriction $E$ increases, the system energy consumption increases and the time delay decreases to a certain extent. This is because APs can consume more energy to process computing tasks faster. Besides, from Figure 6, we can observe that the delay curve under the average time of the system no longer drops significantly when the energy consumption limitation $E$ is large enough. At this time, the method of increasing the energy consumption constraint to obtain a significant reduction in the delay is no longer Works. The reason is that the algorithm has tended to converge.

Figure 8 comprehensively shows the convergence process of running the DCS-OCTO algorithm. It is visible that under four different smoothing coefficients, the algorithm converges to the global optimal solution with more iterations when $\eta = 10^{-3}$. However, it stays at the local optimum several times before obtaining the global optimum. By observe Figure 8, as $\eta$ increases until $\eta = 10^{-2}$, the algorithm has
the least number of iterations to get the global optimum. However, if we continue to increase the value of $\eta$, the algorithm will not reach the global optimum with fewer iterations but will make the algorithm more convergent to the inferior solution.

Figure 9 shows how the CPU clock of the AP for computing tasks changes with its own energy shortage. In fact, its own energy shortage is fundamentally determined by the demand for tasks, service caching and task offloading. From Figure 9, we can observe that all APs have a similar trend, that is, when the energy shortage is large, the CPU clock of APs is smaller; when the energy shortage is small, the CPU clock of APs also has a smaller value. This is because when the energy shortage is too large, the APs do not have enough energy. To avoid excessive energy consumption, the APs will actively reduce their own CPU clock to handle less load, and ultimately achieve the goal of reducing energy consumption. When the energy shortage is small, the APs have enough energy to process tasks faster to reduce the computation delay, so the APs start to increase their CPU clock. At the same time, we can also observe that the CPU clock of the APs has not increased steadily with the increase in energy shortage. The reason is that APs choose to directly offload the computing tasks that are currently unable to handle directly to the remote cloud. And the reason why the CPU clocks of the three APs are different under the same energy shortage is that the APs will communicate with each other to cache more types of services, so that adjacent APs will largely handle different types of task.

VI. CONCLUSION

This paper studies joint service caching and task offloading for a MEC three-tier architecture system with multiple users, multiple edge servers, and multiple remote cloud servers, and proposes an efficient online and decentralized algorithm that can adjust service caching decisions for popular patterns of time and space services. Our algorithm is developed based on Lyapunov optimization and Gibbs sampling. Works online without the need for future information and achieves provable near-optimal performance. Simulation results show that the algorithm effectively reduces system delay while ensuring low energy consumption.

APPENDIX A

PROOF OF THEOREM 1

First, we square the left and right sides of the energy shortage queue $Q_m(t + 1) = \max\{\sum_{m \in \mathcal{M}}(E_m^\prime(b^\prime, d^\prime) + \hat{E}_m^\prime) - E[Q(t)], 0\} + Q_m(t)$ shown in equation (7) and multiply by $\frac{3}{2}$ to get the following equation.

$$\frac{3}{2}Q^2(t + 1) = \frac{3}{2}(\hat{E}^2(b^\prime, d^\prime)^2) + \frac{3}{2}E^2 + \frac{3}{2}Q^2(t)$$

$$+ Q(t) \cdot \hat{E}^2(b^\prime, d^\prime) - Q(t) \cdot E$$

$$- \hat{E}^2(b^\prime, d^\prime) \cdot E$$

(25)

Then, Equation (4-25) is changed to Equation (4-26), that is,

$$\frac{3}{2}Q^2(t + 1) - Q^2(t) = \frac{3}{2}[\hat{E}^2(b^\prime, d^\prime)^2 + E^2]$$

$$+ Q(t) \cdot \hat{E}^2(b^\prime, d^\prime) - Q(t) \cdot E$$

$$- \hat{E}^2(b^\prime, d^\prime) \cdot E$$

(26)

According to Equation (5), $\hat{E}^2(b^\prime, d^\prime)^2 \leq \sum_{m \in \mathcal{M}}(E_m^\max)^2$ can be obtained, and after rounding out the fourth term on the right side of Equation (26) and changing the sum of squares to the sum of squares, we can finally get Inequality (27), i.e.,

$$\frac{3}{2}[Q^2(t + 1) - Q^2(t)]$$

$$\leq \frac{3}{2}[\sum_{m \in \mathcal{M}}((E_m^\max)^2 + E^2)]$$

$$+ Q(t) \cdot \hat{E}^2(b^\prime, d^\prime) - Q(t) \cdot E$$

$$\leq \frac{3}{2}(\sum_{m \in \mathcal{M}}E_m^\max)^2$$

$$+ Q(t) \cdot \hat{E}^2(b^\prime, d^\prime) - Q(t) \cdot E$$

(27)

Therefore, Lyapunov offset $\Delta L(Q(t))$ obviously satisfies the following inequality, which is to obtain the upper bound of $\Delta L(Q(t))$, as shown below

$$\Delta L(Q(t)) \leq \frac{3}{2}(\sum_{m \in \mathcal{M}}E_m^\max + E)^2$$

$$+ Q(t)\mathbb{E}[\hat{E}^2(b^\prime, d^\prime) - E|Q(t)]$$

(28)
Let \( B = \frac{1}{2}(\sum_{m \in M} E_m^{max} - E)^2 \), then formula (28) be transformed into formula (29),

\[
\Delta L(Q(t)) \leq B + Q(t) \mathbb{E}[\bar{E}(b^t, d^t) - E(Q(t))] \tag{29}
\]

Therefore, Theorem 1 is proved.

**APPENDIX B**

**PROOF OF THEOREM 2**

First, we define the Bell number fits recursive formula

\[ \sum_{k=1}^{K} \binom{K}{k} \] corresponding to the service type, from which it is concluded that there are \( X \) cases of service type combinations in the system. Next, the action vector \( \Psi = \psi_1, \psi_2, \ldots, \psi_X \) corresponding to the service cache decision is defined. The vector has \( X \) dimensions, and the service cache decision of AP \( m \) meets \( b'_m \in \Psi \) in any time slot. Then, because there are \( M \) APs in the system, the service cache decision \( b' \) of time slot \( t \) is regarded as a Markov chain of \( M \) dimension, and the service cache decision of AP \( m \) corresponds to the \( m - \text{th} \) dimension of the Markov chain.

For the sake of explanation, it is assumed that there are only two APs in the system, that is, the Markov chain has only two dimensions. Then, the state of the Markov chain in time slot \( t \) is defined as \( S_{b'_1,b'_2} \), and the total state space is defined as \( Z \). Among them, \( S_{b'_1,b'_2} \in Z; b'_i \in \Psi, i = 1, 2, \ldots \). Because in each iteration, selecting an AP from the system to explore new service cache decisions is random, so the probability of each AP being selected is equal. And then get the state distribution, that is,

\[
P(S_{b'_1,b'_2} ; S_{b'_1,b'_2}) = \begin{cases} \frac{-\delta(S_{b'_1,b'_2})}{\eta}, & \text{if } b'_1 = b'_2 \text{ or } b'_2 = b'_1; \\ 2X(e^{\frac{-\delta(S_{b'_1,b'_2})}{\eta}} + e^{\frac{-\delta(S_{b'_1,b'_2})}{\eta}}), & \text{otherwise} \\ 0, & \text{otherwise} \end{cases} \tag{30}
\]

Among them, \( \delta(S_{b'_1,b'_2}) \) represents the value corresponding to the objective function \( P^2 \) when the state of the Markov chain is \( S_{b'_1,b'_2} \). Next, we determine whether the stable distribution is true by checking the meticulous equilibrium conditions, that is, whether Equation (31) is true.

\[
\sum_{x=2}^{X} p^*(S_{\psi_1,\psi_2}) \cdot P(S_{\psi_1,\psi_2} | S_{\psi_1,\psi_1}) = \sum_{x=2}^{X} p^*(S_{\psi_1,\psi_2}) \cdot P(S_{\psi_1,\psi_1} | S_{\psi_1,\psi_1}) \tag{31}
\]

Then, we take Equation (30) into Equation (31), and we get Equation (32)

\[
\sum_{x=2}^{X} p^*(S_{\psi_1,\psi_2}) \cdot e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}} = \sum_{x=2}^{X} p^*(S_{\psi_1,\psi_2}) \cdot e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}} \\
2X(e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}} + e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}}) \tag{32}
\]

From Equation (32), it can be seen that \( \tilde{S}, \tilde{S} \in Z \) is balanced in any state. The smoothly distributed is \( P \ast (\tilde{S}) = \kappa e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}} \), and \( \kappa \) is a constant. According to the law of conservation of probability, the stable distribution of Markov chains in any state \( \tilde{S}, \tilde{S} \in Z \) is finally obtained as shown below

\[
P^*(\tilde{S}) = \frac{e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}}}{\sum_{S_{\psi_1,\psi_1} \in Z} e^{\frac{-\delta(S_{\psi_1,\psi_1})}{\eta}}} \tag{33}
\]

It can be seen that the Markov chain does not have merging and is irreducible, so the stable distribution (33) is effective and unique.

Finally, let \( S^* = \arg \min_{S_{\psi_1,\psi_1} \in Z} \delta(S_{\psi_1,\psi_1}) \), that is, the state of the corresponding Markov chain when \( P^2 \) takes the minimum value. Then \( \lim_{n \to \infty} P^*(S_{\psi_1,\psi_1}) = 1 \) shows that the DCS-OCTO algorithm converges to the optimal state under probability. Similarly, according to the analogy analysis, when the 2-dimensional Markov chain is extended to the \( X \) dimension, the above conclusions are also satisfied.

**APPENDIX C**

**PROOF OF THEOREM 3**

First, we introduce Lemma C.1.

**Lemma 1:** For any \( \theta > 0 \), there is a stable and random strategy for \( P^2 \), which is independent of the current energy shortage queue backlog and determines \( b_i^t, d_i^t \) and thus satisfies the inequality \( \mathbb{E}[\bar{E}(b_i^t, d_i^t) - E] \leq \theta \).

**Proof:** Similar proofs are shown in Theorem 4 and Theorem 5 in [34]. For simplicity, the proof is omitted here. Then, we bring Lemma C.1 into the drift-and-delay expression \( \Delta L(Q(t)) \), that is, Equation (20), we get

\[
\Delta L(Q(t)) + V \cdot \mathbb{E}[\bar{T}(b_i^t, d_i^t) | Q(t)] \\
\leq B + Q(t) \mathbb{E}[\bar{E}(b_i^t, d_i^t) - E | Q(t)] \\
+ V \mathbb{E}[\bar{T}(b_i^t, d_i^t) | Q(t)] \\
\leq B + Q(t) \cdot \theta + V \cdot (T^{opt} + \theta) \tag{34}
\]

The second inequality in Inequality (34) is because strategy \( \Pi \) has nothing to do with the energy shortage queue. Let \( \theta = 0 \), sum the inequality over \( t = 0, 1, \ldots, s - 1 \), and divide the result by \( s \) to get

\[
\frac{1}{s} \sum_{t=0}^{s-1} \mathbb{E}[\bar{E}(Q(t)) - L(Q(0))] + V \sum_{t=0}^{s-1} \mathbb{E}[\bar{T}(b_i^t, d_i^t)] \\
\leq B + V \cdot T^{opt} \tag{35}
\]

Rearrange these terms, taking into account the fact that \( L(Q(t)) \geq 0, L(Q(0)) = 0 \) and finally the left and right sides of Inequality (35) are divided by \( V \). From this, the delay boundary of the system under the average time is obtained, that is, Theorem 3 is proved.

**REFERENCES**

[1] R. Roman et al., “A survey and analysis of security threats and challenges,” Future Gener. Comput. Syst., vol. 78, no. PT.2, pp. 680–698, 2018.
