Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 1548142, 13 pages
https://doi.org/10.1155/2021/1548142

Research Article

A Novel Multitask Scheduling and Distributed Collaborative Computing Method of Edge Nodes in the Internet of Things

Yong Wang,[1] Siyu Tang,[2] Xiaorong Zhu,[2] and Yonghua Xie[1]

[1]*Nanjing Vocational University of Industry Technology, Nanjing 210003, China*
[2]*Nanjing University of Posts and Communications, Nanjing 210003, China*

Correspondence should be addressed to Xiaorong Zhu; 2018100906@niit.edu.cn

Received 30 April 2021; Revised 29 July 2021; Accepted 1 December 2021; Published 31 December 2021

Academic Editor: Xin Liu

In this paper, we propose a novel multitask scheduling and distributed collaborative computing method for quality of service (QoS) guaranteed delay-sensitive services in the Internet of Things (IoT). First, we propose a multilevel scheduling framework combining the process and thread scheduling for reducing the processing delay of multitype services of a single edge node in IoT, where a preemptive static priority process scheduling algorithm is adopted for different types of services and a dynamic priority-based thread scheduling algorithm is proposed for the same type of services with high concurrency. Furthermore, for reducing the processing delay of computation-intensive services, we propose a distributed task offloading algorithm based on a multiple 0-1 knapsack model with value limitation with the collaboration of multiple edge nodes to minimize the processing delay. Simulation results show that the proposed method can significantly reduce not only the scheduling delay of a large number of time-sensitive services in single edge node but also the process delay of computation-intensive service collaborated by multiple edge nodes.

1. Introduction

With the rapid development of the IoT applications, the number of devices connected to the network has increased dramatically, the volumes of data have an explosive growth, and there have been higher computing and storage requirements on the centralized approach of task processing, as represented by cloud computing, whereas edge computing is a distributed architecture that features decentralized processing and storage power, where the central node and the edge node collaborate to complete the computing process, dispersing the storage and computation tasks of the central node to the edge of the network, making full use of the equipment resources and reducing the pressure of computing and storage on the central node. It is predicted more than 70% of data will be analyzed, processed, and stored at the edge side in the future, which can reduce delay and improve response time to meet the delay requirements in IoT scenarios [1]. The implementation system of edge computing architecture in IoT is called edge nodes (ENs), which may be gateways with intelligent computing functions, smart routers, or other embedded devices. ENs serve for different types of services with different QoS requirements such as ultradelay, high bandwidth, or high reliability. They are now becoming increasingly popular for IoT applications, such as the Internet of Vehicles (IoV) scenario, where many ENs for data processing are deployed on both sides of the highway in order to minimize the processing delay of data. Therefore, the diversity characteristics of IoT applications bring many challenges to EN. Especially, how to guarantee the QoS of delay-sensitive services is a hot research topic, and in recent years, many literatures have proposed some effective solutions, mainly including multitask scheduling algorithms [2–8] and collaborative computing algorithms [9–15].

Multitask scheduling means that multiple processes can run simultaneously in a computer system, and multiple threads can also run simultaneously within each process. The reasonable multitask scheduling including the combination of process and thread can serve more services and simultaneously meet the delay requirements of different services. In [2], a multitask scheduling problem on parallel machines was investigated and the objective was to

minimize the weighted sum of the earliness and tardiness. In [3], the authors propose different scheduling strategies for different tasks based on VxWorks in an embedded real-time operating system area. In order to reduce memory consumption in real-time multitasking, [4] proposed an improved real-time scheduling algorithm based on Least Laxity First (LLF) to reduce maximal heap memory consumption by controlling multitask scheduling, without modifying the processing order of each task. [5] studied an extension of the completely fair scheduler (CFS) to support cooperative multitasking with time-sharing for heterogeneous processing elements. It shows that cooperative multitasking is possible on heterogeneous systems and it can increase application performance and system utilization. The authors in [6] introduced an ordinal optimization algorithm using rough models and fast simulation to obtain suboptimal solutions in a much shorter timeframe. In [7], a task scheduling model of computational grid was built and a particle swarm algorithm was to solve the task scheduling problem. In [8], the authors proposed the composite scheduling for multitask algorithm which defines the task priority as its own unique priority level for each task and assigns the same priority to the multiple tasks if a number of tasks are introduced in scheduling.

On the other hand, the execution time of computation-intensive services would be long if carried out by only EN due to its limited computational capability. To shorten the execution time of the task, it is optional to offload the task to other neighbour ENs for assisted computing. In [9], the authors studied the subcarrier and power allocation problem of the Orthogonal Frequency Division Multiple Access (OFDMA) system to minimize the maximal delay of each mobile device by offloading computationally intensive tasks to the ENs provided by the cellular base stations. [10] presented a task offloading policy taking into account execution, energy consumption, communication overhead, and other expenses. In [11], a communication-constrained mobile edge computing (MEC) framework and an optimal task scheduling policy based on Lyapunov theory were developed, respectively. However, the above-mentioned schemes only consider the offloading of computing load between two nodes. [12] considered an OFDMA-based multiuser and multi-MEC-server system, where the task offloading strategies and wireless resources allocation were jointly investigated. [13] gave a Cooperative Computing System (CCS) framework and designed the task migration algorithm to achieve execution reliability for the highest priority task in the distributed computing environment. In [14], the power splitting model was proposed. In this model, the transmission signal of an IoT node is divided into two power streams, and the power factors are allocated according to the difference in signal power. Therefore, the IoT node can transfer 5G and IoT information with different power streams. [15] proposed the method of maximizing the transmission rate of each node through the power allocation proportion to ensure the power optimization of node.

Therefore, the existing studies usually consider fewer factors and fail to provide an effective way for multitask scheduling and computational collaboration scheme for delay-sensitive service of IoT. Hence, in this paper, for guarantying the QoS of delay-sensitive services, we first propose a centralized multitask scheduling scheme within an EN and then give the distributed collaborative computing scheme for computational intensive task among multiple ENs. The contributions of our works are as follows:

(1) A centralized multitask scheduling method is proposed in the SEN scenario to the problem of how to reduce the processing delay of a large number of time-sensitive tasks in IoT. This method first designs a multilevel scheduling framework which combines process and thread scheduling. The process scheduling uses a preemptive static priority scheduling algorithm for different kinds of services, and the tread scheduling is based on dynamic priority for the same kind of service with high concurrency. This algorithm is based on the task execution urgency factor which is determined by the remaining computation within the task deadline

(2) A distributed collaborative computing method is proposed in the MEN scenario to the problem of how to reduce the processing delay of computation-intensive tasks in IoT. This method first designs a distributed collaborative computing framework based on the asynchronous message queue which enables multiple nodes to serve the same task collaboratively through splitting the task. Then, a task offloading decision algorithm based on a multiple 0-1 knapsack model with value limitation is designed to minimize the task processing delay with the optimal offloading scheme

2. System Model

The system model in this paper is shown in Figure 1, where the aggregation nodes collect the data from neighbouring sensors and forward it to the ENs which carry out computing and scheduling functions for different services. In this paper, we focus on two types services: time-sensitive and computation-intensive services. Here, time-sensitive service may include emergency alarm class, control command class, session class, and cycle monitoring class according to different priorities. For this kind of service, there may be a large number of tasks at the same time and the complexity of the scheduling algorithm will be high. Therefore, it needs to minimize the task processing delay by an effective multitask scheduling strategy within a single edge node (SEN). Whereas for computation-intensive service, the computation of a single task is relatively large and the processing time of a SEN is too long, it is necessary to reduce the processing delay of the task by distributed collaborative computing with multiple edge nodes (MENs). Therefore, it needs to design a distributed collaborative computing method to reduce the processing delay.

2.1. Multitask Scheduling Framework of Time-Sensitive Services. Here, we propose a hierarchical multitask scheduling framework combining process and thread scheduling, as shown in Figure 2. The framework opens a process for

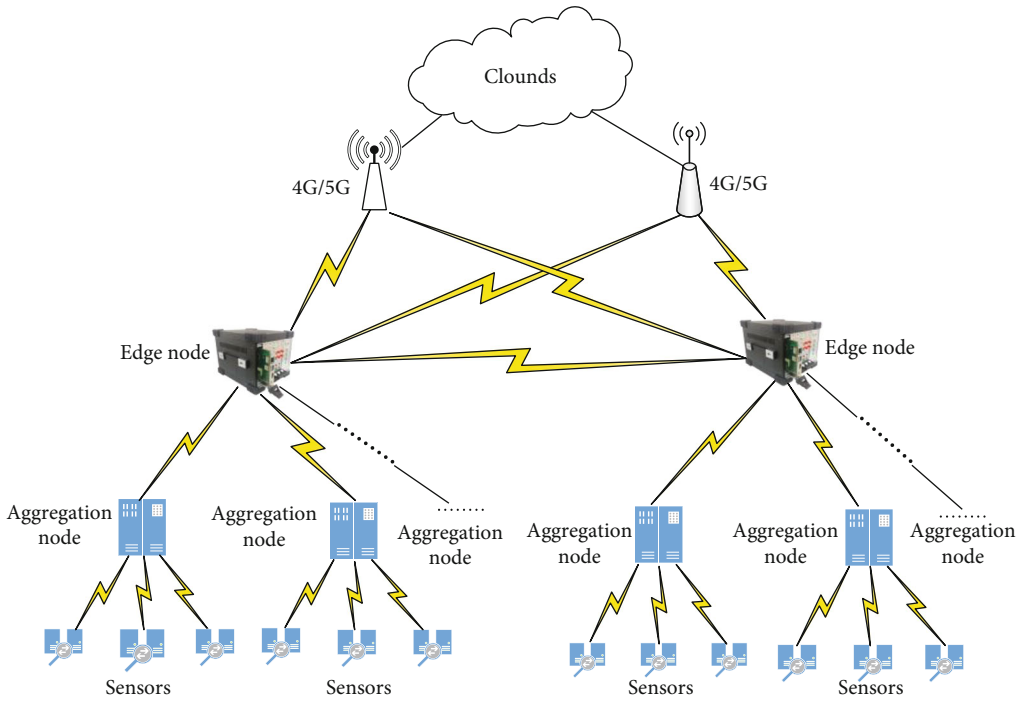

Figure 1: Schematic diagram of the system model.

each IoT service, and multiple tasks of the same service are scheduled as threads within the corresponding process. That is, process scheduling is applied to different types of IoT services, and thread scheduling is applied to different tasks of the same services. Therefore, the hierarchical scheduling is like the relationship between processes and threads, where the processes dominate the threads. The proposed hierarchical scheduling framework is described as follows:

(1) When a task arrives, the service admission and classification mechanism in the edge node identify which type of IoT services the task belongs to

(2) Once the type of the task is determined, the REQ packet of the task is sent to the REQ queue of the corresponding services process. The REQ packet of each task contains the data needed for the task processing

(3) After the service process scans for a new task in its REQ queue, it creates a new thread to process the task. In this service process, the scheduling framework executes a multithread scheduling algorithm

(4) The scheduling framework executes a multiprocess scheduling algorithm for different service processes

(5) After a task in the service process is successfully executed, the RESP packet of this task is fed back to the RESP queue of the service process, and the RESP packet contains the returned value of the successful execution of the task

(6) The corresponding user terminal (UE) receives the RESP packet, and the task ends

2.2. Distributed Collaborative Computing Framework of Computation-Intensive Services. For computation-intensive tasks, we design a distributed collaborative computing architecture based on asynchronous message queue for MENs, which can offload tasks that are difficult to execute by a single node to multiple neighbour nodes. Furthermore, based on the proposed collaborative computing framework, we propose an optimal task offloading algorithm to minimize the time-consuming execution of the original tasks. The asynchronous message queue uses messages to connect different programs. It is a component based on the Point to Point (P2P) messaging model; i.e., a message can only be used by a consumer only once, and the mechanism of prevention of repeated consumption of messages is quite perfect. In a distributed collaborative computing scenario, a message is the data content of communication between programs, and it contains three elements:

(1) Message producers: create the carrier of the message, which can generate messages and send messages to the message queue.

(2) Message queues: memory area for storing messages is the channel between the different programs. The messages are stored in order according to the order of receiving message from message producers.

(3) Message consumers: a carrier extracts messages from message queues and consumes them. Message consumers generally have more than one.

As shown in Figure 3, in the MEN scenario, the message producer is the central edge node (CEN), and the message consumer is the neighbour edge node (NEN) around the

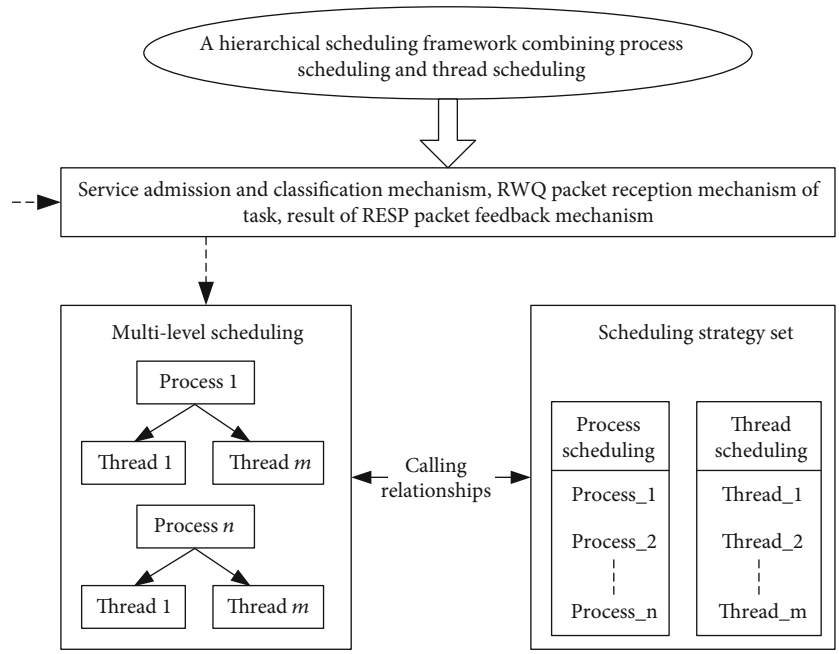

Figure 2: A scheduling framework combining process scheduling and thread scheduling.

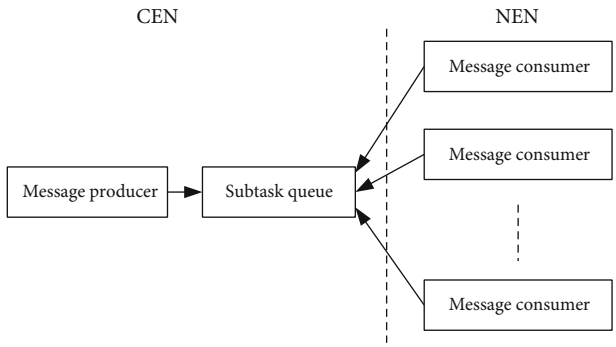

Figure 3: Architecture of asynchronous message queue for P2P messaging model.

CEN. When the CEN produces user tasks (original tasks) beyond the scope of its own computing capabilities, it can be divided into multiple subtasks (Sub-Task) based on the properties of the original task itself. At this point, the original task is transformed into a task set (Task Set) containing multiple subtasks. The CEN pushes the subtasks in the task set into the subtask queue (Sub-Task Queue), from the NEN to the task queue stored in the shared memory to pull the task and execute the task; after the execution is completed, the result file of the subtask is feedback and summarized to CEN.

The implementation platform of distributed collaborative computing system is accomplished by building a CPU cluster system. The CPU cluster system considers the CPU cores, i.e., the computing resources distributed on each MEN, as the carriers for subtask on MENS. The computing resources allocated to subtasks which are pulled from the task queue by MEN are in terms of the number of CPU cores. Building a CPU clustering system is a common choice solution for high-performance computing architecture. It will not only speed up the original task computation but also be more flexible. It can allocate subtasks to specified computing resource blocks on specified MEN in the most effective way. The functional design diagram of MEN-distributed collaborative computing framework is shown in Figure 4.

The functional implementation flow is shown in Figure 5.

Step 1. The user or the terminal generates a task T. The amount of computation contained in the original task causes a large computational load on the CEN. According to the properties of the original task T, the CEN transforms the task T into a task set containing several subtasks. The more original tasks T generated by the user, the more task sets on the CEN.

Step 2. For each task set generated on CEN, all the information (including subtask file paths, the estimated time of subtasks, the size of hard disk space occupied by subtasks files, etc.) of the subtask in this task set is packaged and serialized and pushed into the task queue one by one;

Step 3. The task offload module in CEN selects the appropriate NENs for the subtasks in the task queue through the allocation algorithm (presented in Section 3); then, the correspondence between subtasks and NENs was established. It should be noted that if there are no remaining computing resources in each NEN (all CPU cores are occupied) at this time, then, the remaining subtasks in the queue will be in a waiting state. Once there are free computing resources in the NEN, the subtask offloading will be continued to execute;

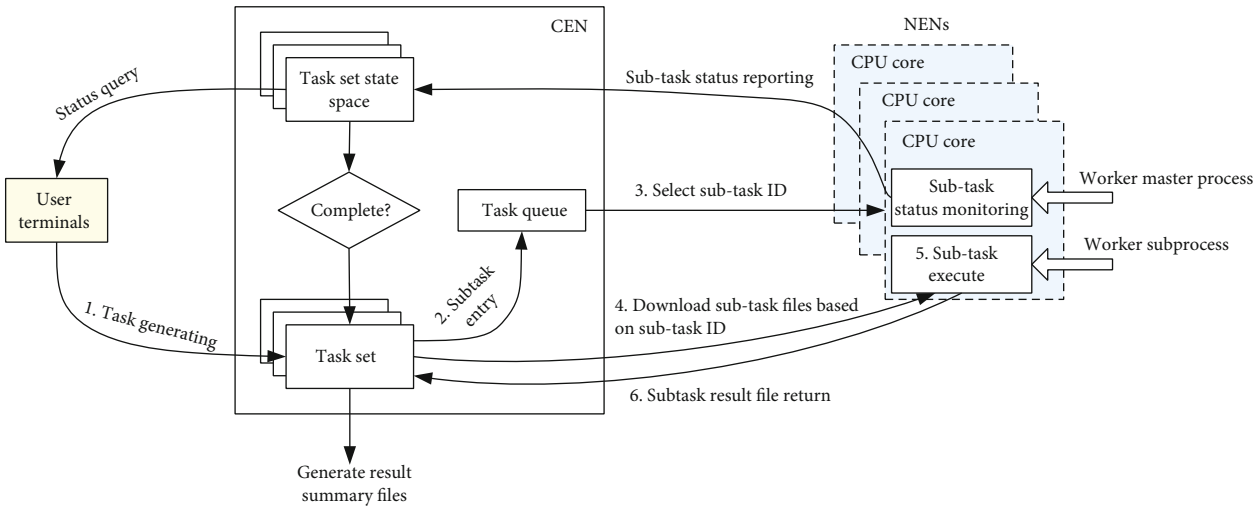

Figure 4: Functional design diagram of MEN-distributed collaborative computing framework.

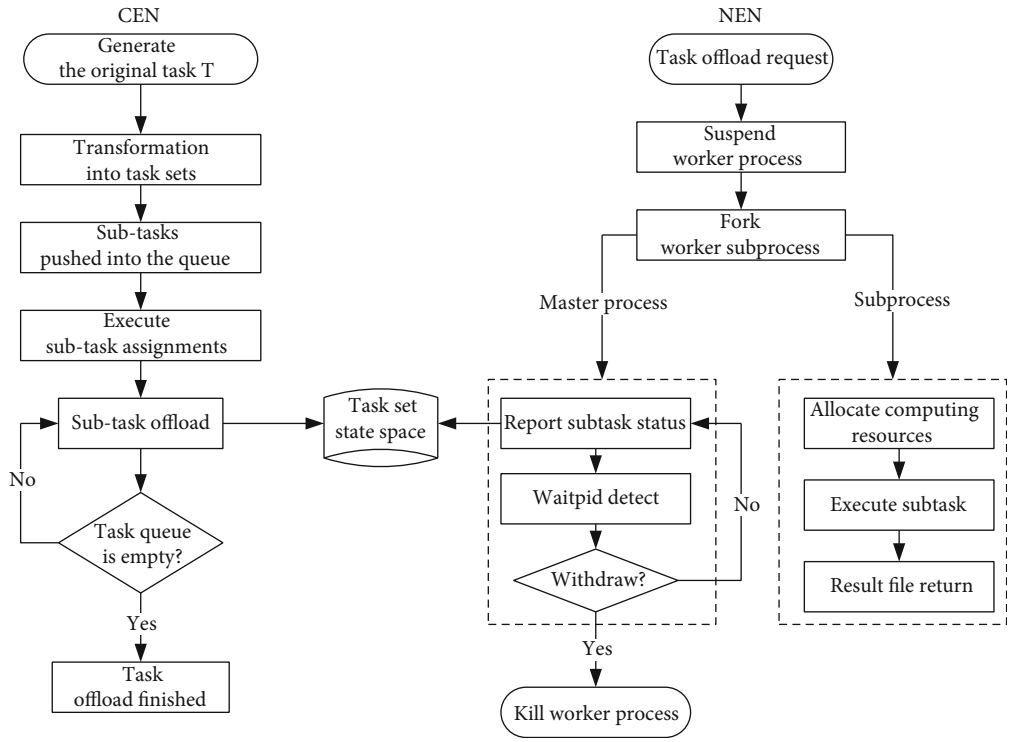

Figure 5: Functional flow chart of MEN-distributed collaborative computing system.

Step 4. The task assignment module sends the subtask ID to the corresponding NEN through socket and downloads the subtask file from the CEN by local.

Step 5. After NEN downloads the subtask file, the Master process of NEN starts a Worker process for the subtask and forks a Worker subprocess in the Worker master process. As shown in Figure 5, the Worker subprocess is responsible for the execution of the subtask, and the Worker master process monitors the status of the Worker subprocess (i.e., the execution status of the subtask) by waitpid and reports the monitored status information to the status space of task set in CEN periodically.

The execution of the subtask in the Worker subprocess includes two steps. First, the computational resource area is divided for the execution of subtask; that is, the CPU cores are allocated to the Worker subprocess through the taskset binding tool of the Linux kernel to ensure the computational resources occupied by the subtasks during execution are fixed. Second, execute the subtask.

Step 6. The subtask is executed, and the result file is returned.

Step 7. CEN checks the state space of taskset to ensure the execution status of all subtasks in the task set. If it shows that the execution of all subtasks in this task set has been successful, CEN starts to summarize the result files of the subtasks in the task set; otherwise, it continues to wait.

3. Multitask Scheduling Algorithm of SEN

In this section, we first propose a multiprocess scheduling algorithm based on preemptive static priority and then a multithread scheduling algorithm based on dynamic priority to reduce the processing delay of a large number of small tasks with time-sensitive requirement in IoT scenarios.

3.1. Multiprocess Scheduling Algorithm Based on Preemptive Static Priority. In the service type classification of IoT, each service has strict requirements on processing delay; this means that when the execution of multiple service process conflicts, the system will inevitably give the CPU processing right to the service process with high priority. Therefore, the scheduling algorithm of preemptive static priority is adopted in the scheduling strategy of multiple processes.

The different service processes are given different priorities by the static priority scheduling. The priority will not change during the system operation; that is, the system always executes the service process with higher priority. In an actual environment, the low-priority periodic monitoring service occurs more frequently, so the system runs this service process most of the time. High priority services such as emergency alarms occur relatively infrequently, but once they occur, the system will abandon the processing of the other types of services and give the CPU processing right to the alarm service process. Preemptive scheduling is that the system processes the service process strictly in accordance with the service priorities. When the high priority comes, the system will prioritize all current tasks and start execution from the highest priority task.

For example, when the edge nodes use the Linux kernel, the priority of the process is determined by two factors: one is the PRI value of the process, and the other is the NI value of the process. The true priority of the process is determined by the sum of the PRI value and the NI value. The PRI value of the process cannot be modified, but the priority of the services process can be configured by modifying the NI value. Before this, the Process ID (PID) of the services process is obtained, and then modify the service priority though renice tool: renice new_ni PID. In an actual test, it only needs to modify the NI values of different services processes to make the comprehensive priority level meets the division method of IoT services, so that the purpose of preemptive scheduling can be achieved.

3.2. Multithread Scheduling Algorithm Based on Dynamic Priority. In the multilevel scheduling framework combining process scheduling and thread scheduling in the SEN scenario, process scheduling uses a preemptive static priority scheduling algorithm and thread scheduling is applied to different tasks of the same service. This section designs a scheduling algorithm based on dynamic priority for multithread scheduling, which achieves asynchronous IoT task processing and also ensures the processing timeliness of delay-sensitive services.

The dynamic priority scheduling algorithm is very different from the static priority algorithm. The static priority means that the process with a fixed number of priorities at the beginning of its generation and the priority does not change during the process running, whereas dynamic priority changes the priority of the process during the process running to get the good scheduling performance. Clearly, it is a more flexible and customizable scheduling strategy that prevents a process from occupying the CPU processing power of the system for a long term. This is crucial for the processing of asynchronous tasks in IoT. Dynamic priority scheduling also has some disadvantages. For example, with the change of priority, the context switching of the process will consume certain system performance. However, this section applies the idea of dynamic priority scheduling of processes to thread scheduling, and the impact of context switching of threads on system performance is negligible.

The dynamic priority algorithm proposed in this section is not the traditional preemptive dynamic priority scheduling. The algorithm chooses a time slice as the scheduling cycle. Therefore, every thread can be allocated an execution time in the scheduling cycle, which is more suitable for processing asynchronous IoT tasks.

The key element of dynamic priority is the setting of real-time dynamic priority P_{dyn} of thread. Suppose that the set of tasks $\text{TS} = \{t_1, t_2, \cdots, t_m\}$ consists of m tasks in service process. For each task in TS, the REQ package contains at least two kinds of information: the estimated execution time T_C and the reasonable delay T_D (processing delay that does not affect the quality of user). The performance of the dynamic priority scheduling algorithm is determined by the relationship between the actual execution time T_R and the reasonable delay T_D of the task.

The requirements of delay of tasks are not much different in the same service process. The setting of the real-time dynamic priority P_{dyn} of the task thread cannot be determined by the estimated execution time T_C or the reasonable time delay T_D of the task, because the urgency of the task cannot be determined by the one value of T_C or T_D. The calculation formula of the urgency factor λ of the task is defined as

$$\lambda = \begin{cases} \dfrac{T_{\text{RC}}}{T_{\text{RD}}}, & \text{if } T_{\text{RD}} > T_{\text{RC}}, \\ 1, & \text{else}, \end{cases} \tag{1}$$

where T_{RC} is the remaining execution time of the task, which is calculated by the difference between the estimated execution time T_C of the task and the executed time of the task. T_{RD} is the remaining reasonable delay of the task, which is calculated by the difference between the reasonable delay T_D of the task and the time the task has reached the

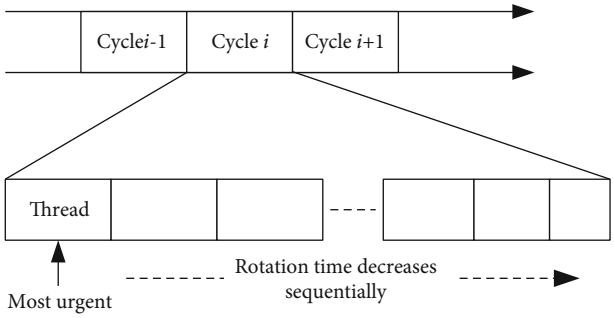

FIGURE 6: Diagram of rotation time allocation in a scheduling cycle.

service process. When $T_{RD} \leq T_{RC}$, it means that the execution of the task may exceed the deadline, and λ must be set to 1, which represents the highest priority of task. The larger the value of λ_i of task i, the higher priority of task i; then, the longer rotation time slice of task i will be allocated in the next service process scheduling cycle T_{DC}.

Define the weight of the rotation time of task i in the next scheduling cycle T_{DC} of the service process as W_i, and the calculation formula is given by

$$W_i = \frac{\lambda_i}{\sum_{j=1}^{m} \lambda_j}. \tag{2}$$

Therefore, the rotation time T_i allocated to task i in the next scheduling cycle T of the service process is calculated by

$$T_i = W_i \bullet T_{DC}. \tag{3}$$

The rotation time weight W_i of task i represents the real-time dynamic priority P_{dyn} of the thread during the scheduling cycle. In the next scheduling cycle, the system will prioritize the task thread with high real-time dynamic priority P_{dyn} and allocate the previously calculated rotation time to it, as shown in Figure 6.

In this algorithm, the setting of the scheduling period T_{DC} is also critical. The size of T_{DC} will affect the effect of the entire scheduling algorithm. If T_{DC} is too small, it means that the switching frequency of multiple task threads in the business process will be very high. Although the cost of thread switching is small, such high-frequency context switching will still bring greater performance loss to the system. If T_{DC} is too large, the effect of the scheduling algorithm under coarse-grained granularity will be discounted. With reference to the Complete Fair Scheduling (CFS) algorithm in Linux, the scheduling cycle is 2 ms-6 ms. This algorithm also sets the scheduling cycle to 6 ms, which will be verified in the following simulation.

In summary, in the multilevel scheduling framework of SEN scenarios, the process of multithread scheduling algorithm based on dynamic priority is as follows:

Step 1. The service process detects whether there is a new task in the REQ queue and, if so, obtains the estimated execution time T_C and the reasonable delay T_D of the task in the REQ package and opens a new task thread for it and, if not, jumps to Step 2.

Step 2. Update the task set TS, the remaining execution time T_{RC}, and the remaining reasonable delay T_{RD} of all tasks in TS.

Step 3. Calculate the urgency factor λ of all tasks in the task set TS using formula (1).

Step 4. Use formulas (2) and (3) to calculate the rotation time weight W and rotation time T of all tasks in the task set TS in the next scheduling period T_{DC}, and configure W as the real-time dynamic priority P_{dyn} of the task thread.

Step 5. The system starts scheduling according to the real-time dynamic priority P_{dyn} of the task thread;

Step 6. Jump to Step 1;

4. Distributed Collaborative Computing Algorithms of MEN

In the functional process of the distributed collaborative computing system in the MEN scenario, the subtask offloading module in Step 3 is equated to a multiple 0-1 Knapsack Model based on Value Limitation (VLKM), by solving the optimal unloading strategy to further reduce the execution time of the original task. In the MEN scenario, the factors that influence the offload strategy need to be considered and are shown in Table 1.

The task offloading decision model is shown in Figure 7, where n edge nodes are deployed in the MEN scenario, and these nodes constitute a set of edge node NENs, NENs = $\{\mathrm{NEN}_1, \mathrm{NEN}_2, \cdots, \mathrm{NEN}_n\}$. At this moment, there is an edge node as CEN to generate an original task T and transform T into a task set $\mathrm{TS} = \{\mathrm{st}_1, \mathrm{st}_2, \cdots, \mathrm{st}_m\}$ containing m subtasks. The data transmission rate between CEN and NEN_k is c_k.

The subtask factors to be considered for multiedge node computing collaboration include the following: the estimated execution time T_{pi} and the computing resource requirement C_{pi} of subtask i and the storage resource requirements S_p of the subtask. Since the number of CPU cores of the NEN is limited and C_p is related to T_p, the C_p of the subtask can be set to 1 core and 2 cores according to the relationship between the average estimated execution time T_{avg} of the subtask in T_p and TS.

$$T_{avg} = \sum_{i=1}^{m} T_{pi},$$
$$C_{pi} = \begin{cases} 1, & T_{pi} \leq T_{avg}, \\ 2, & T_{pi} > T_{avg}. \end{cases} \tag{4}$$

The factors of NEN that need to be considered for multiedge node computing collaboration include the following:

Table 1: System parameters.

Parameter type	Parameter	Symbol	Unit
Subtask related	Estimated execution time of subtasks	T_P	Second
	Subtask computing requirements	C_P	CPU core
	Subtask storage requirements	S_P	MB
Network environment related	Subtask file synchronization time-consuming	T_d	Second
NEN related	NEN remaining computing resources	C_{idle}	CPU core
	NEN remaining storage resources	S_{idle}	MB

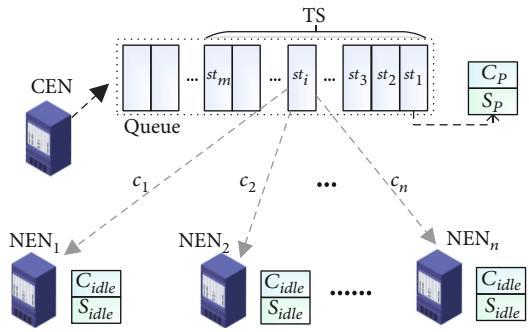

Figure 7: Task offloading decision model diagram.

the remaining computing resources C_{idle} and the remaining storage resources S_{idle}; this information is uniformly reported to CEN before the offloading decision is executed.

The optimal offloading strategy should be able to make full use of the computing resources and storage resources of NEN. Therefore, the decision of subtasks offloading can be transformed into a backpack model: there are several items, each item has two attributes, volume C_p and value S_p, and the volume of the backpack is C_{idle}; the target is to find a solution that makes the value of the items in the backpack maximum.

The maximum number of each subtask in the task queue which can be offloaded to NEN_k $(1 \le k \le n)$ is 1, so this is another 0-1 knapsack problem. In this backpack model, items represent subtasks, and the volume of the backpack represents the margin of computing resources on the NEN_k.

In the task offloading decision model, the solution of the knapsack problem must consider the trade-off between the estimated execution time T_p of the subtask and the synchronization time T_d of the subtask folder; that is, the length of the execution time of the subtask should be complementary to the length of synchronization time of the subtask fold. Before finding the optimal solution for the knapsack model, a method of preprocessing is as follows:

Step 1. Consider the estimated execution time T_p of the subtasks: sorting the subtasks in the task set TS according to the estimated execution time T_p in a descending order; that is, the subtask that takes a long time is offloaded firstly.

Step 2. Consider the synchronization time T_d of the subtask folder: sorting the neighbour edge nodes in NENs according to the transmission rate between edge nodes and CEN in a descending order; that is, the subtasks of the NEN with a higher transmission rate are offloaded with higher priority, and this strategy uses the least time-consuming synchronization.

According to these two preprocessing steps, the subtasks that are expected to take longer time to execute will be offloaded first and will be offloaded to the NEN that takes the least synchronization time to execute. So, the trade-off between the estimated execution time T_p of the subtask and the synchronization time T_d of the subtask folder is considered.

Any kind of knapsack problem can be transformed into a general 0-1 knapsack; the multiple 0-1 knapsack problem with value limitation in this model can be transformed into multiple general 0-1 knapsack problems with a limited value. The optimal solution for general 0-1 knapsack model is to obtain the value matrix:

$$\text{Value} = \begin{bmatrix} 0 & \cdots & \\ \vdots & \ddots & \vdots \\ \cdots & a & \end{bmatrix}. \tag{5}$$

The element Value$[i][j]$ in the value matrix represents the maximum value of the first i items in the case of the knapsack capacity with j. And the method for solving Value$[i][j]$ is expressed by

$$\text{Value}[i][j] = \max\left(\text{Value}[i-1][j], \text{Value}[i-1][j-w[i]] + v[i]\right), \tag{6}$$

where $w[i]$ represents the volume of item i and $v[i]$ represents the value of item $i, i \ge 1, j \ge 1$. Under the conditions of N items and the knapsack capacity of W, the maximum value generated by the knapsack problem is Value$[N][W] = a$.

In the task offloading decision model, a subtask represents an item, which has two attributes: the calculation requirement C_p of the subtask and the storage requirement S_p of the subtask. The current edge node NEN_k represents a backpack, which has two attributes: the remaining computing resources C_{idle} and the remaining storage resources S_{idle}. The value matrix Value in the 0-1 knapsack problem

is transformed into the storage matrix Mem in the task unloading decision model. In a real-world scenario, multiple subtask files unloaded on NEN_k are hardly the same size as the remaining storage resource S_{idle} for NEN_k. Hence, when the size of these unloaded subtask files reaches a threshold of S_{th}, NEN_k will no longer be assigned new subtasks. The calculation of threshold S_{th} is divided into two cases:

When the storage resources of NEN_k are quite sufficient, that is, the maximum value of the storage matrix is $Mem[m][C_{idle}] < S_{idle}$; S_{th} is set as

$$S_{th} = \alpha Mem[m][C_{idle}](\alpha \leq 1). \quad (7)$$

When the storage resources of NEN_k are insufficient, that is, the maximum value of the storage matrix is $Mem[m][C_{idle}] > S_{idle}$; S_{th} is then set as

$$S_{th} = \alpha S_{(idle)}(\alpha \leq 1). \quad (8)$$

In each value limitation 0-1 Knapsack model, it is firstly assumed to be a normal 0-1 knapsack problem. After obtaining the storage matrix Mem, the previous value $Mem[x][y]$ that is first larger than S_{th} is found by horizontal traversal of the matrix to limit the total size of the offloaded subtask files, x represents the offloading scheme selected by NEN_k from the previous x subtasks in the task queue, and y represents the size of the computing resources occupied by the subtasks in this offloading scheme.

The horizontal traversal is designed to minimize the value of x. Then, the subtasks offloaded to NEN_k are selected from among the subtasks that are expected to take longer to execute. Referring to the first two preprocessing steps, the purpose of the horizontal traversal is to fully consider the equilibrium between the expected execution time T_p of subtasks and the synchronization time T_d of subtask files. After obtaining $Mem[x][y]$, the backtracking algorithm can be used to derive which subtasks of the previous x subtasks of the task queue are offloaded to NEN_k for execution. At this point, the solution of the regular 0-1 backpacking problem with value constraint for NEN_k is completed.

The subtasks that are offloaded to the edge node NEN_k in the task set TS are removed, and the offloading policy analysis is continued for NEN_{k+1} until the task set TS is empty. At this point, the optimal subtask offloading policy for the MEN scenario can be derived.

Therefore, the subtask offloading decision algorithm based on multiple 0-1 knapsack model based on value limitation can be expressed as follows:

Step 1. Preprocess:

(a) Sorting the subtasks in the task set TS according to the estimated execution time T_p in descending order

(b) Sorting the NEN in the edge node set NEN_s according to the transmission rate c_i between the edge nodes and the CEN in a descending order

Table 2: Hardware parameters.

Parameter type	Specific parameters
SOC	Broadcom BCM2711
CPU	64 bit 1.5 GHz 4 cores (28 nm)
GPU	Broadcom VideoCore VI@500 MHz
Bluetooth	Bluetooth 5.0
USB interface	USB2.0∗2/USB3.0∗2
HDMI	Micro HDMI*2 support 4K60
Power	Type C (5V 3A)
WIFI	802.11AC wireless 2.4 GHz/5 GHz double frequency
Wired Ethernet	Gigabit Ethernet

Step 2. Initialize $\alpha = 0.9$, k from 1 to n iteration:

(a) CEN obtains the remaining computing resources C_{idle}, remaining storage resources S_{idle}, and data transmission rate of CEN in NEN_k, and CEN reports to the Offload Decision Center (ODC) together with the computing requirements C_p and storage requirements S_p of all subtasks in the local task set TS; then, the ODC makes the offload strategy

(b) In ODC, the storage matrix Mem of NEN_k is obtained through Equation (6) of value matrix Value

(c) Traverse the storage matrix Mem horizontally, and find the previous value $Mem[x][y]$ that is first greater than the S_{th} value

(d) Use the three parameters of x, y, and $Mem[x][y]$ to find the subtasks contained in $Mem[x][y]$ through the backtracking algorithm, label these subtasks k, and push the subtasks into the task queue in the order of labeling

(e) Clear the subtasks in the task set TS that have been labeled in Step (d), and check whether the task set TS is empty; if it is empty, skip the iteration; otherwise, continue the iteration

Step 3. The ODC sends the offload strategy back to CEN, and CEN executes the offload operation for the subtasks.

Step 4. Check whether the task set TS is empty; if it is empty, the algorithm ends; otherwise, wait for the node in the NENs to release the resource, and jump to Step 1.

5. Performance Evaluation and Analysis

In order to evaluate the performances of proposed algorithms, we have built the experimental evaluation system, where we deploy four edge nodes with the Linux operating system and the CPU is BCM2711 chip with 1.5 GHz 4-core of Broadcom. The programming language is C++. The edge nodes are the new Raspberry Pi 4th with Linux system, and the main hardwares of parameters are shown in Table 2.

5.1. Performance Evaluation of Proposed Multitask Scheduling Algorithm. In this part, we make performance evaluation of proposed multitask scheduling algorithm. Since the processing time of emergency alarms and control instructions is very short and the frequency of services is not very high, we focus on the delay evaluation of conversational services in this simulation system. Here, we assume the processing delay requirement is between 500 ms and 2 s. In addition, in order to verify the effect of the algorithm in delay-sensitive service processing, the average waiting time T_{wait} and the standard deviation σ_{wait} of the waiting time are used as indicators. The average waiting time is the average of the actual processing time T_R of each task exceeding the reasonable delay T_D. If the task is processed successfully before the reasonable delay requirement, then, the T_{wait} of this task is 0. The standard deviation of the waiting time is the standard deviation of the waiting time of the same batch of tasks, which reflects the statistical dispersion of the waiting time of the task and reflects the algorithm causing the waiting time of some tasks to be too long.

As a comparison of the effects of the proposed dynamic priority scheduling algorithm (multithread DPS) in this paper, this paper also implements an improved completely fair scheduling algorithm (CFS) and a shortest job priority algorithm (SJF). CFS assigns a fixed rotation time to tasks in the scheduling cycle according to the preset task weights set; that is, the time slice size of the task is fixed in different scheduling cycles, and the priority of the CFS algorithm is determined by the ratio of the task running time to the task weight. In other words, the CFS algorithm depends on the preset task weights and lacks the flexibility relatively. Since CFS is a process scheduling algorithm, the weight of the process depends on the NI value of the process. The factor for setting task weights is the reasonable time delay of the task; that is, the higher the task delay requirement, the higher the task weight. The SJF algorithm transfers the CPU processing power to the process (thread) with the shortest execution time. If the execution time of the two tasks is the same, then, the task can be executed according to the First Come First Served (FCFS).

In the simulation, three algorithms are applied to the same batch of tasks, respectively, and the performance of the algorithm is judged by comparing the size of the average waiting time T_{wait} and the standard deviation σ_{wait} of the waiting time. The smaller T_{wait} and σ_{wait}, the better the algorithm performs. In order to highlight the effect of the algorithm, the verification system binds service processes to a single CPU core, that is, multiple task threads are scheduled on one CPU core of a certain EN. In actual verification, a batch of tasks is first generated according to the delay index of the conversational services. The main parameters of the tasks include the following: the estimated executed time of task T_C and the reasonable delay of the task T_D. The estimated execution time of the task is less than the reasonable delay of the task (500 ms-2 s). The average estimated processing time of this batch of tasks is 95.7 ms, and the average reasonable delay is 1156.7 ms. The algorithm executes 20 task threads, and Table 3 shows the relationships between scheduling period and average waiting time. It can be seen from the table that it is reasonable that the scheduling cycle is set as 6 ms which can maximally improve the performance of the algorithm and has minimal influence on the

Table 3: Relationship between scheduling period and average waiting time.

Scheduling period	Average waiting time
4 ms	79.2 ms
6 ms	80.0 ms
10 ms	83.1 ms
20 ms	91.4 ms
50 ms	120.5 ms
80 ms	151.4 ms

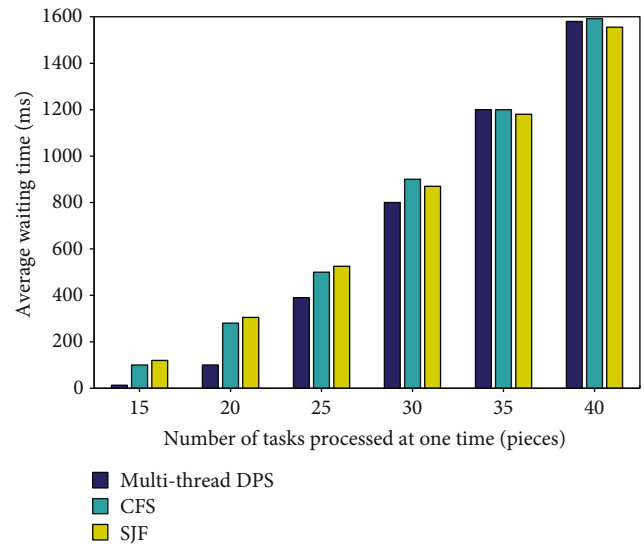

Figure 8: Comparison of average waiting time of tasks of three algorithms.

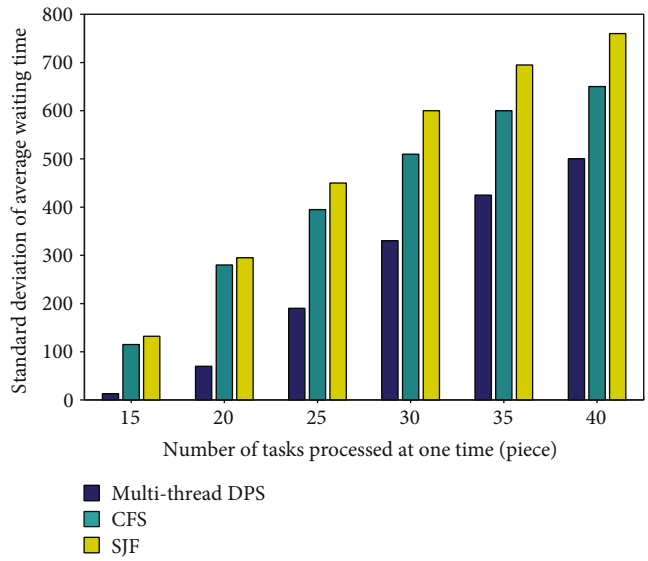

Figure 9: Comparison of standard deviation of waiting time of task for three algorithms.

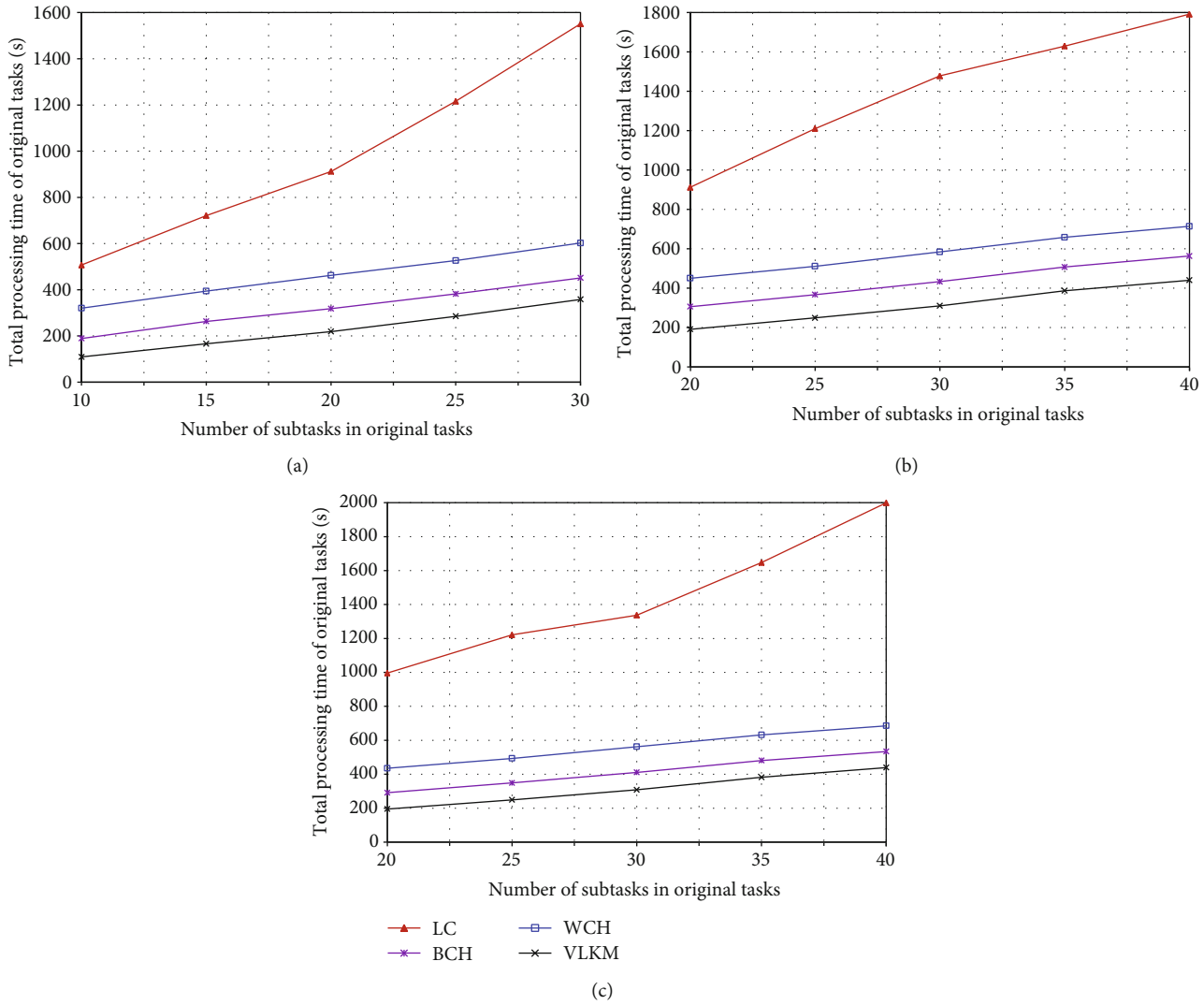

FIGURE 10: The total processing time with different number of subtasks. (a) The collaborated number of ENs is 10, and the number of CEN is edge node 1. (b) The collaborated number of ENs is 15, and the number of CEN is 1. (c) The collaborated number of ENs is 15, and the number of CEN is 5.

performance when the scheduling period fluctuates in a small range around 6 ms.

Figure 8 shows the change of average waiting time of three algorithms with different numbers of tasks processed at one time with the scheduling period of 6 ms. It can be seen that the performance of the proposed multithread DPS algorithm is significantly better than the other two algorithms when a small number of tasks are executed on single CPU core of the edge node. The core of the SJF algorithm is to ensure the priority of tasks with short execution time; that is, the estimated execution time of the task is considered. The weight setting of the CFS algorithm depends on the processing delay requirements of the task, and the higher the delay requirements of tasks, the higher the weights of tasks. Therefore, the SJF algorithm and the CFS algorithm only consider the estimated execution time T_C and the delay requirement T_D of task, respectively. When the amount of task gradually increases, the performance of the three scheduling algorithms shows no difference. This is an inevitable result indicating that the amount of the tasks has reached the limitation of a single CPU core.

The proposed multithread DPS algorithm comprehensively considers the estimated execution time of the task and the delay requirement of the task, and the most urgent task can always be executed first in the next scheduling cycle. Therefore, the proposed multithread DPS algorithm can not only reduce the average waiting time but also reduce the standard deviation of the waiting time and prevent the processing delay of some tasks from being too long. Figure 9 shows the comparison of standard deviation of waiting time of task for three algorithms. It can be seen that the standard deviation of task waiting time of the CFS algorithm and the SJF algorithm is relatively large. The consequence is that the processing delay of some tasks is relatively small, but the processing delay of other tasks is particularly long. Regardless of the load of the edge node, the proposed multithread DPS algorithm has a smaller waiting time standard deviation

than the other two algorithms, and hence, there is seldom excessive processing delay of some tasks.

5.2. Performance Evaluation of Distributed Collaborative Computing Algorithm. In order to evaluate the proposed multiple 0-1 knapsack model based on value limitation (VLKM) for distributed collaborative computing algorithm, we compare it with the local calculation model (LC), the optimal model of consistent hashing (Best Consistent Hash (BCH)), and the worst model of consistent hashing (Worst Consistent Hash (WCH)). The local computing model is a single-node task computing, and the other three algorithms are multinode collaborative computing solutions.

We assume an EN captures a video through a camera, and now, it needs to transmit this video to a specified user for play. If it is directly transmitted, the transmission time is longer, and more users need to transmit and each user's cell phone resolution is different. Therefore, the video needs to be processed into video streams with different resolutions on EN, which is more suitable for network transmission. This current EN acts as the CEN at this moment. The CEN transforms the original task T into task set which contains 24 subtasks. Each subtask file contains a subsequence of the original video yuv sequence and a video encoding algorithm Algorithm_n (i.e., subtask script). The function of the subtask is to generate a video stream of each yuv sequence with different resolutions which is suitable for network transmission by the subtask script. The result file returned by the subtasks is the data monitored during the encoding process. After all the tasks in the task set are executed successfully, then, CEN will start the collection of all subtask files of this task set.

Figure 10 shows the relationship between the total processing time and the number of subtasks of original task under different deployments of ENs and CNs. In Figures 10(a) and 10(b), there is a CEN and the number of deployed ENs is 10 and 15, respectively. In Figure 10(c), there are 15 ENs and 5 CNs. The simulation results show that the performance of the proposed VLKM is always better than those of the other three algorithms in the MEN environment, which can significantly decrease the processing delay of the original task.

6. Conclusion

This paper first designs a multilevel scheduling framework that combines process scheduling and thread scheduling and proposes a preemptive static priority scheduling algorithm and a thread scheduling algorithm based on dynamic priority. Simulation results show that the proposed multitask scheduling strategy has a significant effect on time-sensitive processing of a large number of small tasks in SEN scenario. Then, a collaborative computing framework among MEN based on asynchronous message queue is designed, and a task offloading decision algorithm based on the multiple 0-1 knapsack model based on value limitation is proposed. Simulation results show that this algorithm can significantly minimize the processing delay of large tasks. In our future work, we will further study the collaborative computation algorithm among wireless ENs for industrial Internet of Things.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61871237 and 62076123), the Key R&D Plan of Jiangsu Province (BE2021013-3), and the Foundation of Nanjing Institute of Industry Technology (YK1802012).

References

[1] Gartner, Inc, *Hype Cycle for Edge Computing*, 2019, http://www.gartner.com.

[2] M. Liu, R. Liu, and X. Liu, "Two-stage stochastic programming for parallel machine multitasking to minimize the weighted sum of tardiness and earliness," in *2019 16th International Conference on Service Systems and Service Management (ICSSSM)*, pp. 1–6, Shenzhen, China, July 2019.

[3] R. Peng and X. Zheng, "A multitask scheduling algorithm for VxWorks: design and task simulation," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, pp. 353–357, Shanghai, China, November 2009.

[4] Y. Machigashira and A. Nakata, "An improved LLF scheduling for reducing maximum heap memory consumption by considering laxity time," in *2018 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 144–149, Guangzhou, China, 2018.

[5] T. Beisel, T. Wiersema, C. Plessl, and A. Brinkmann, "Cooperative multitasking for heterogeneous accelerators in the Linux Completely Fair Scheduler," in *ASAP 2011-22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 223–226, Santa Monica, CA, USA, 2011.

[6] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya, "Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 338–351, 2014.

[7] H. Li, L. Wang, and J. Liu, "Task scheduling of computational grid based on particle swarm algorithm," in *International Joint Conference on Computational Science and Optimization*, pp. 332–336, Huangshan, China, 2010.

[8] S. A. Hussain, C. S. Ramaiah, J. Chinna Babu, and M. N. Giri Prasad, "IOT multitasking: design of smart phone application for systematic execution and scheduling in real time environment," in *MEC International Conference on Big Data and Smart City (ICBDSC)*, pp. 1–6, Muscat, Oman, 2019.

[9] M. Li, S. Yang, Z. Zhang, J. Ren, and Y. Guanding, "Joint subcarrier and power allocation for OFDMA based mobile edge computing system," in *2017 IEEE 28th Annual International*

Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), pp. 1–6, Montreal, QC, Canada, 2017.

[10] Q. Zhu, B. Si, F. Yang, and Y. Ma, "Task offloading decision in fog computing system," *China Communications*, vol. 14, no. 11, pp. 59–68, 2017.

[11] X. Yang, Z. Chen, K. Li, Y. Sun, and H. Zheng, "Optimal task scheduling in communication-constrained mobile edge computing systems for wireless virtual reality," in *2017 23rd Asia-Pacific Conference on Communications (APCC)*, pp. 1–6, Perth, WA, Australia, 2017.

[12] K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, "Energy-efficient joint offloading and wireless resource allocation strategy in multi-MEC server systems," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, Kansas City, MO, USA, 2016.

[13] M. Hasan and M. S. Goraya, "A framework for priority-based task execution in the distributed computing environment," in *2015 International Conference on Signal Processing, Computing and Control (ISPCC)*, pp. 155–158, Waknaghat, India, 2015.

[14] X. Liu and X. Zhang, "Rate and energy efficiency improvements for 5G-based IoT with simultaneous transfer," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 5971–5980, 2019.

[15] X. Liu, X. Zhai, W. Lu, and C. Wu, "QoS-guarantee resource allocation for multibeam satellite industrial Internet of Things with NOMA," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 3, pp. 2052–2061, 2021.