Hindawi
International Journal of Aerospace Engineering
Volume 2024, Article ID 9922714, 17 pages
https://doi.org/10.1155/2024/9922714

Research Article

A Dynamic Task Scheduling Algorithm for Airborne Device Clouds

Bao Deng[1,2] **and Zhengjun Zhai**[1]

[1]*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China*
[2]*Xi'an Aeronautics Computing Technique Research Institute, Aviation Industry Corporation of China, Xi'an 710068, Shaanxi, China*

Correspondence should be addressed to Bao Deng; bs1706002@sust.edu.cn

Received 17 August 2023; Revised 17 December 2023; Accepted 18 January 2024; Published 26 February 2024

Academic Editor: Zhiguang Song

The rapid development of mobile Internet has promoted the rapid rise of cloud computing technology. Mobile terminal devices have greatly expanded the service capacity of mobile terminals by migrating complex computing tasks to run in the cloud. However, in the process of data exchange between mobile terminals and cloud computing centers, on the one hand, it consumes the limited power of mobile terminals, and on the other hand, it results in longer communication time, which negatively affects user QoE. Mobile cloud can effectively improve user QoE by shortening the data transmission distance, reducing the power consumption, and shortening the communication time at the same time. In this paper, we utilize the property that genetic algorithm can perform global search seeking the global optimal solution and construct a dynamic task scheduling model by combining the device-cloud link. The task scheduling model based on genetic algorithm and random scheduling algorithm is compared through comparison experiments, which show that the assignment time of the task scheduling model based on genetic algorithm is shortened by 11.82% to 48.51% and the energy consumption is reduced by 22.28% to 47.52% under different load conditions.

1. Introduction

Accompanied by the rapid popularization of the 5G network, the booming development of cloud computing technology, and people's extensive use of mobile terminals, including cell phones, iPads, tablets, drones, and the Internet of Things (IoT), cloud computing [1] has gained explosive development in the field of mobile Internet, in particular mobile cloud computing (MCC). Mobile terminals are playing an increasingly important role in people's lives. However, mobile devices have limited computing power and cannot perform operations such as complex data processing and large-scale computing. Mobile cloud computing (MCC) is an emerging cloud service model that allows users to connect their mobile devices to mobile cloud servers to accomplish various tasks or leave them to a central cloud to perform complex computing tasks.

By establishing a connection between the mobile device and the cloud, the task is given to the cloud through the mobile network, and when the cloud finishes executing the task, the execution result is returned to the terminal through the mobile network [2, 3]. With the increasing complexity of tasks to be handled by mobile devices and the increasing dependence of users on mobile devices, reducing the energy consumption of mobile devices to extend the usage time has become an important research direction. In the process of submitting tasks and receiving execution results from the cloud, mobile devices need to consume a lot of power. The current mobile terminal battery storage capacity is generally small, and frequent data exchange will significantly shorten the use of mobile devices. Paczkowski [4] pointed out that the short battery life is the most prominent point that affects the iPhone user experience. To address these problems, researchers have proposed the concept of "cloudlet." A

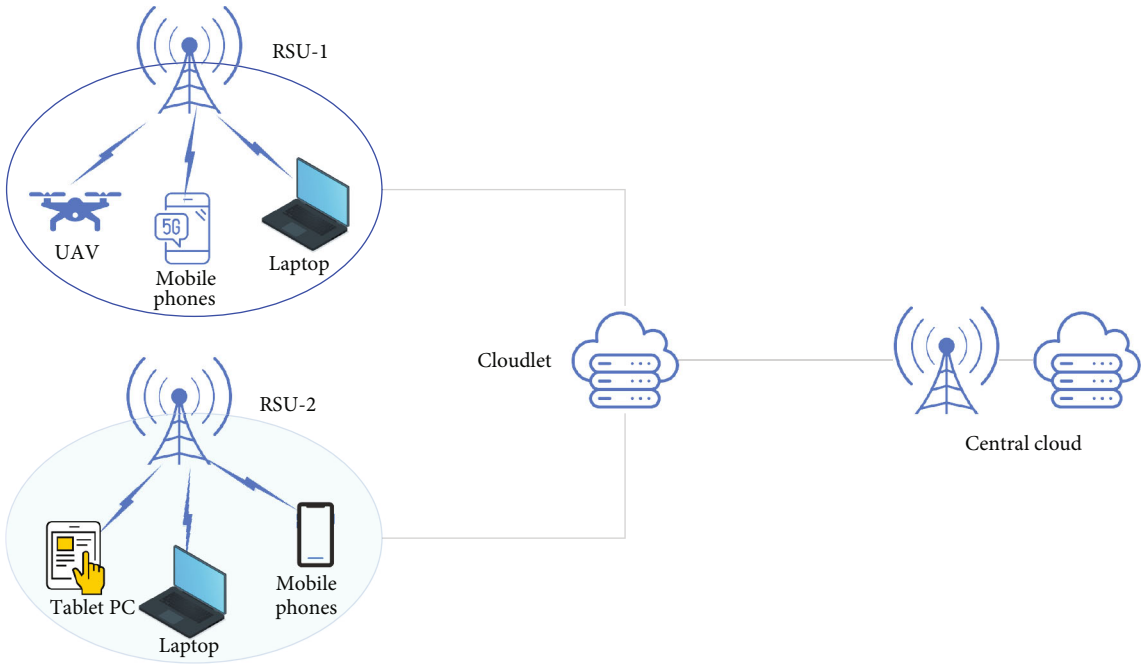


Figure 1: Mobile device cloud architecture.

cloudlet is a small, trusted cloud with computing power that can be connected to a nearby mobile device. Such a cloudlet can minimize the power consumption required for task transfer and can reduce the time delay [5]. At the same time, handing over tasks or data to a microcloud is fully achievable for the user because of the computational and storage capabilities of the microcloud. Efficient embedded task scheduling algorithms are more capable of improving operational efficiency and reducing the energy consumption of microclouds [6, 7]. The mobile device cloud architecture is shown in Figure 1.

UAV technology has made rapid development in recent years, and related research for UAV on-board equipment cloud has become a hotspot. Loke [8] provides services to mobile users based on airborne equipment and optimizes the configuration of airborne equipment and ground facilities according to the problems arising from different scenarios to provide the best QoS and QoE, reliability, scalability, etc. According to Yang et al. [9], to balance the load of a multi-UAV-assisted mobile edge computing (MEC) system, a multi-UAV deployment mechanism based on differential evolution (DE) is proposed, which models the access problem as a generalised assignment problem (GAP) and then solves the problem with a near-optimal solution algorithm, and based on this, a deep reinforcement learning (DRL) algorithm is proposed for UAV task scheduling, which improves the UAV task execution efficiency. Xie et al. [10] proposed a geometry-based layout algorithm to generate the optimal layout position of UAVs to achieve more energy-efficient task scheduling. A low-complexity divide-and-conquer scheme was proposed for the nonconvex task scheduling and resource allocation problem, which decomposed the original problem into three subproblems to solve them separately. Extensive simulation results show that the framework has good energy efficiency. Zhou et al. [11] design a task scheduling strategy to minimize the offloading and computational delays of all tasks given the UAV energy capacity constraints. The online scheduling problem is first formulated as an energy-constrained Markov decision process (MDP), while a new deep risk-sensitive reinforcement learning algorithm is developed to assess the risk of each state, and a large number of simulation results show that the algorithm reduces the task processing latency by 30% compared to the probabilistic configuration method while satisfying the UAV energy capacity constraint.

The development of IoT technology has enriched the microcloud application scenarios, and the access of diverse sensors and end devices puts forward a severe test for the microcloud's task scheduling capability. To solve the above problems, the main contributions of this paper are as follows.

(i) Constructs a dynamic scheduling model for microcloud tasks by taking advantage of the characteristics of the overall exploration strategy of the genetic algorithm and the optimized search method that does not rely on gradient information or other auxiliary knowledge during computation, but only on the objective function and the corresponding fitness function that affects the search direction

(ii) Determines the optimal parameters of the model in the MDC environment through experiments

(iii) The performance of the dynamic task scheduling model based on a genetic algorithm is verified through experiments

2. Related Works

Along with the development of cloud computing, its powerful storage capacity as well as computing power provides services for more and more users. And in recent years, with the rapid development of mobile devices, mobile cloud computing has become the focus of researchers. Many applications require a relatively large amount of computation, so if they are run on mobile devices, it may lead to a device system performance bottleneck [12]. Based on this, researchers have come up with the concept of mobile cloud computing; based on the mobile cloud model, if a mobile device user wants to use this type of application, he or she can give the computational task to the cloud to perform. Other applications such as image retrieval, speech recognition, gaming, and navigation applications can be run on the mobile device system; however, they consume a relatively large amount of power, and therefore, it is not a good option to leave them to be executed by the mobile device itself.

The explosive development of mobile devices, especially the extremely rapid development of UAVs, has provided a broad application prospect for mobile device cloud. Sun et al. [13] proposed a new big data framework that exploits the parallel processing capability of cloud computing to process large-scale remote sensing data while incorporating task scheduling strategies to further exploit the parallelism of the distributed processing stage. The approach first analyzes remote sensing applications and characterizes them as directed acyclic graphs (DAGs). The obtained DAG is used to represent the application to develop an optimization framework that combines distributed computing mechanisms and task scheduling strategies to minimize the total execution time. By determining an optimal scheme for task partitioning and task allocation, high utilization of cloud computing resources can be achieved, which significantly improves the speed of remote sensing data processing. Zhou et al. [14] first formulated the MCPS security maximization problem as a mixed integer nonlinear programming (MINLP) problem and then proposed a decomposition algorithm to derive the optimal task scheduling solution without degrading performance transforming it into a mixed integer linear programming (MILP) problem. The derived task scheduling solution determines the allocation of all tasks, the frequency of operation, the order of execution, and the selection of security services. Simulation results show that the system security level of this solution is improved by an average of 20.38% and 65.11% when compared to the existing and baseline approaches. Kim et al. [15] proposed a collaborative task scheduling approach for IoT-assisted edge computing, where the edge node decides where to offload edge tasks among participating IoT devices based on offload execution time and energy consumption, and each IoT device decides when to execute the offloaded tasks considering local task execution. Experimental results show that it outperforms other scheduling algorithms in terms of deadline satisfaction for time-critical tasks.

Scholars have carried out in-depth research on cloud task scheduling for airborne mobile devices from multiple perspectives using a variety of advanced techniques. Pandit et al. [16] proposed a task scheduling system based on a two-stage neural network (NN), in which the first stage consists of a feed-forward neural network (FFNN) and convolutional neural network (CNN) that decides whether the data streams can be analyzed (executed) in the resource-constrained environment (edge/fog) or directly forwarded to the cloud. In a resource-constrained environment (fog) to be analyzed (executed) or forwarded directly to the cloud, the second stage consists of the RL module scheduling all the tasks sent by the first-stage neural network to the fog layer in the available fog devices. Experimental results show that the combination of RL and task clustering algorithm significantly reduces the communication cost. Yang et al. [17] proposed a task scheduling algorithm considering the reliability of equilibrium tasks based on a simplified model, a mathematical tool based on the game theory work, and a task scheduling model for computing nodes. In the cooperative game model, the game strategy is used in the computation of the rate allocation strategy of the task on the node. The analysis of the experimental results shows that the algorithm has better optimization results. Lakhan et al. [18] designed a dynamic application-partitioning workload task-scheduling-secure (DAPWTS) algorithm framework, which consists of a minimum cut algorithm, searching for nodes, energy-enabled scheduling, fault scheduling, and a security scheme, to minimize the node's energy consumption, and securely minimum cut algorithm to divide the application into local nodes and edge nodes. Simulation results show that DAPWTS outperforms the existing baseline approach by 30% in terms of energy consumption, deadline, and application failure in the system. Du et al. [19] proposed an efficient tactical edge mobile cloud service model to solve the problems of military operations, limited device access, lack of edge tactical mobility, and edge information fusion processing capabilities in the tactical edge network environment. The model can provide flexible tactical edge information exchange and information processing capabilities. It is adaptive to the frontline battlefield environment in terms of collaborative sensing, decision-making, time delay, and energy requirements.

Numerous scholars have achieved fruitful results by applying a variety of techniques in their research on task scheduling for mobile devices. In this paper, we decompose the device tasks by level, then use a genetic algorithm for initial task allocation, and finally schedule each one-way task to finally achieve the goal of high efficiency and low energy consumption task scheduling.

3. Methodology

3.1. Overall Research Program. In a mobile device cloud environment, a group of mobile devices is highly collaborative in executing tasks, and the system cannot operate normally if any of the mobile devices run out of power. To ensure that the system execution time is as long as possible, it is necessary to coordinate the execution of tasks among devices, to achieve a balanced consumption of energy by each device and minimize the overall energy consumption while improving the throughput of task execution. On this

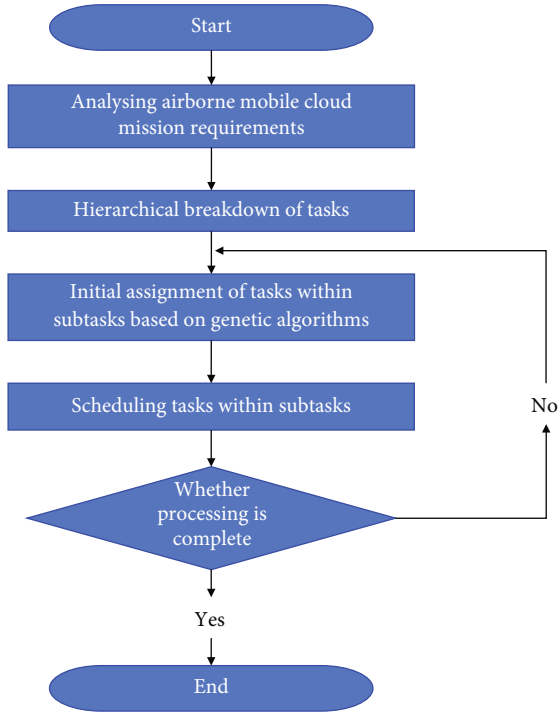


Figure 2: Overall program flow chart.

basis, the corresponding system model is constructed to realize the transformation from practical problems to mathematical problems.

Based on the analysis of the characteristics of the mobile device cloud environment, the mobile device cloud architecture is divided into three levels, i.e., the central cloud, the microcloud, and the mobile device cloud. Each device can be connected to different levels of clouds to communicate with each other and transfer tasks to accomplish the overall task. Since mobile devices in the same mobile device cloud environment are working together to accomplish tasks, it is necessary to process the group tasks into a series of ordered subtasks before assigning them to improve the efficiency of task execution.

Before task scheduling, the purpose of effectively improving the efficiency of selecting schedulable objects can be realized by defining a handshake protocol. To improve the task throughput rate and reduce energy consumption as much as possible, this paper adopts a genetic algorithm for the initial task scheduling algorithm and proposes a dynamic scheduling strategy for the MDC environment in the subsequent scheduling problems caused by the changes of the task or the location of the mobile device. Based on the above framework, this paper proposes to solve the initial task scheduling algorithm based on genetic algorithm and dynamic task scheduling algorithm for mobile devices in the cloud environment of mobile devices. The overall task flow chart is shown in Figure 2.

3.2. Initial Task Scheduling Algorithm Based on Genetic Algorithm. Genetic algorithm has a strong global search ability, can explore the entire problem space, through iteration

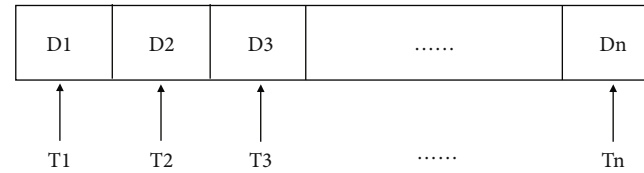


Figure 3: Mapping of task scheduling scheme to chromosomes.

to find the global optimal solution, and has a strong parallelism ability, for distributed edge computing equipment has good adaptability and is very suitable for edge computing equipment task scheduling field. The initial task scheduling algorithm is designed based on the genetic algorithm, because the genetic algorithm has higher efficiency and better convergence and can find an optimal scheduling and allocation scheme in a shorter time, and the genetic algorithm is relatively simple.

Genetic algorithms are capable of evaluating multiple solutions in the exploration space at the same time, reducing the risk of falling into local optima, while the algorithms themselves are easy to parallelize. It is also self-organizing, self-adaptive, and self-learning. When the genetic algorithm uses the information obtained from the evolutionary process to self-organize the search, the individual with a large degree of adaptation has a higher probability of survival and obtains a genetic structure that is more adapted to the environment, which improves the algorithm's robustness. Genetic algorithms provide a framework for the exploitation of optimal solutions for realizing dynamic task scheduling.

3.2.1. Mapping of Task Scheduling Scheme to Chromosome. The coding scheme of the chromosome represents the mapping relationship from the task scheduling scheme to the chromosome, and this paper defines the mapping relationship as follows: the length of the chromosome is the number of tasks, in which each element corresponds to a task, and the value of each element represents the number of the mobile device to which the task is assigned, as shown in Figure 3, which shows the mapping from the task scheduling scheme to the chromosome.

Where the chromosomes can be represented as an array, the subscript of the array corresponds to the number of the task, i.e., T_i, and the value of the array element is the number of the mobile device, i.e., D_i, to which the task T_i is assigned. Generating an initial population means randomly generating a certain number of chromosomes, which means that tasks are randomly assigned to mobile devices according to probability. For example, when assigning 10 subtasks to 3 devices, if after population initialization, a chromosome of {0, 1, 2, 2, 1, 2, 0, 1, 0, 2} is generated, it means that the task T_1 is executed on the mobile device D_1.

3.2.2. Design of the Evaluation Function of the Fitness Value of Chromosomes. Adaptation degree is based on a specific evaluation standard set for a certain problem; this standard is used to measure the advantages and disadvantages of each individual and then according to the standard filter out the better individual, continue to inherit, and finally get the

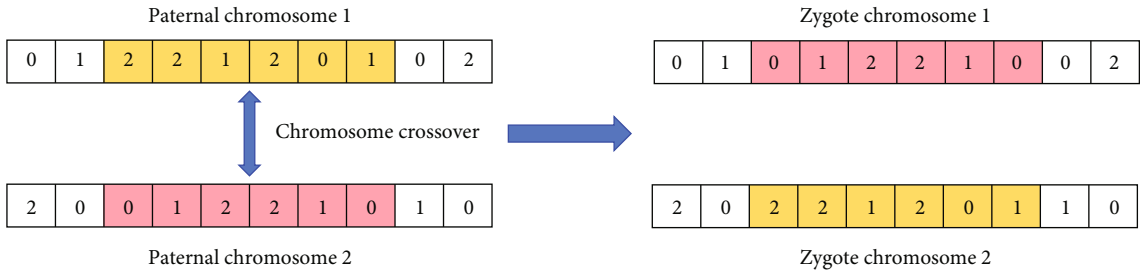


Figure 4: Schematic diagram of chromosome crossover.

optimal solution. The fitness function is transformed from the objective function, and in this paper, the objective function is defined by the dual criteria of time and cost. We define the objective function as follows:

$$f(x) = \alpha * \text{Total\_time} + \beta * \text{Total\_energy}, \quad (1)$$

where Total_time is the total time spent to perform all the tasks resulting from the computation and Total_energy is the total energy spent by each device to perform all the tasks and the sum of the transmission energy required to assign the tasks. α and β correspond to the percentage of time and energy spent, respectively,$\alpha + \beta = 1$.

The scheduling goal in this paper is to minimize the objective function, so the objective function $f(x)$ number is transformed into the fitness function $F(x)$, and the fitness value function is shown in the following equation:

$$F(x) = \begin{cases} C_{\max} - f(x), & f(x) < C_{\max}, \\ 0, & \text{else}, \end{cases} \quad (2)$$

where $C_{\max}$ is a sufficiently large constant. Since our goal is to find the solution that makes the fitness function as large as possible, we need to subtract the value of the objective function from a sufficiently large constant to achieve the goal of finding a better solution.

3.2.3. Design of Chromosome Crossover and Mutation. The initial operation of chromosome crossover is to select genetically good individuals, which will be subjected to crossover operation so as to obtain better offspring individuals. In this paper, the OX crossover operator [20] is used to randomly determine the crossover locations of the parent chromosomes and to determine the lengths of the gene segments to be exchanged based on a predetermined crossover rate, which is then interchanged to generate two daughter chromosomes. Next, the fitness values of the daughter chromosomes are calculated, and if the daughter individual is superior to the parent individual, i.e., the fitness value of the daughter chromosome is greater than that of the parent chromosome, then the parent individual is removed from the population and the daughter individual is added; and if the parent individual is superior to the daughter individual, then the fitness values of the daughter individual and the worst individual (i.e., the individual with the smallest fitness value) in the population are compared. If the offspring individual is superior, the worst individual is removed from the population, and the offspring individual is added; otherwise, the parent individual is reselected for this round of crossover operation.

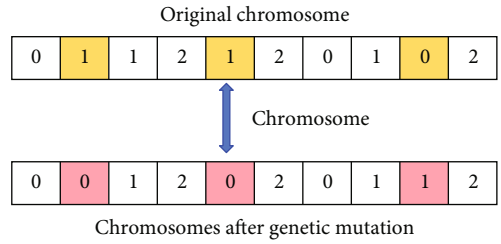


Figure 5: Schematic representation of chromosome variation.

As shown in Figure 4, when the crossover rate is 0.6, the paternal chromosomes cross over to produce the chromosomes of the offspring.

Chromosome mutation also comes from the principle that when organisms reproduce, the genes of the parent generation are mutated, leading to further development of population diversity and better evolution. In genetic algorithms, the chromosome mutation operation can improve the local search ability of the algorithm and, to a certain extent, effectively solve the problem of early convergence, which is conducive to the maintenance of chromosome diversity. The scheduling algorithm based on the genetic algorithm in this paper is essential to migrate a subtask to another device for execution.

In this paper, the random mutation method is used, i.e., according to the mutation rate, certain bits of the chromosome are randomly selected and randomly mutated to become a legal genetic bit, i.e., the number of the device to which a certain task is assigned. Figure 5 shows the schematic diagram of the chromosome after mutation of the original chromosome when the mutation rate is 0.3.

3.2.4. Design and Implementation of Initial Task Scheduling Algorithm Based on Genetic Algorithm. As mentioned above, the design of the initial task scheduling algorithm can be completed. The termination conditions of the algorithm are (1) the algorithm finds a feasible solution and the scheduling ends normally and (2) the number of invalid iterations of the algorithm exceeds the preset maximum allowable number of iterations, forcing it to terminate and taking the

INPUT: Data related to subtask structure, equipment, task energy time consumption, etc.
OUTPUT: The optimal solution obtained from the calculation
Begin
1. calculate the priority of each node in the graph
2. Generate the initial solution group $Pop(t)(t = t)$
3. Calculate the fitness value of each solution in the cluster.
4. if the termination condition of the algorithm is not satisfied, perform step 5, otherwise go to step 9.
5. using the idea of evolutionary strategy, perform the selection mechanism to form the next generation of solutions $Pop(t + 1)(t = t + 1)$
6. perform hybridization with probability p_c
7. perform the mutation operation with probability p_m
8. Calculate the fitness value of each solution in the cluster, and use the elite strategy to preserve the optimal solution.
9. output the best solution, the algorithm terminates
End

Algorithm 1: Initial task scheduling algorithm based on genetic algorithm.

final iteration solution as the optimal solution. The algorithm description is shown in Algorithm 1.

In the initial stage of the scheduling algorithm, the structure graph is first processed and the priority of each node, i.e., its depth, is calculated. The initial population is generated based on each initial information. The initial population is generated based on each initial information. The specific process of initial population generation is that the tasks are first divided into different subsets according to their depth (priority), and the number of subsets is $h + 1$, where h is the maximum value of the node depth in the task structure graph. The nodes with depth i are in the subset $S(i)$ $(0 < i < h)$, and for each subtask in the subset $S(i)$, the assigned mobile device number is randomly assigned, which means that the task is randomly assigned to a mobile device, which corresponds to the result that the value corresponding to each locus of the chromosome is one of the middle $1 \sim m$, where m represents the total number of mobile devices.

As mentioned above, after randomly assigning tasks to mobile devices, there is a queue of subtasks on each mobile device, and each subtask corresponds to a priority level; for each device, its task scheduling policy is that the subtasks it owns are listed in order of priority, and the tasks with high priority are executed first. After processing all the tasks on each mobile device, a task scheduling policy, i.e., chromosome, is obtained. By performing the above operations multiple times, multiple chromosomes are obtained and the initial population can be formed.

After generating the initial population, it is necessary to calculate the fitness value of each solution according to the fitness formula and output the optimal solution if it is judged to be a feasible solution; if it is not a feasible solution, it is necessary to generate the next-generation population according to the selection strategy and then carry out the hybridization and mutation operations with the hybridization probability and the mutation probability, respectively, to allow the population to have more possibilities, which is conducive to finding a feasible solution more quickly. The above operations are repeated until an optimal solution is found or the number of evolutionary generations exceeds a set threshold, and the algorithm ends.

The genetic algorithm can perform a global search to seek the global optimal solution, but when the task size grows, the exploration space rapidly expands causing the genetic algorithm to need more time to complete the global search. And when the task size continues to expand, the time complexity of the algorithm increases dramatically [21], so it is difficult to adapt to task scheduling tasks with huge task sizes using this algorithm alone.

4. Dynamic Task Scheduling Algorithm for Mobile Devices

For each level of subtasks, when the initial scheduling of the task is complete, the execution of the task begins. However, during task execution, task rescheduling is required for mobile devices due to the following situations:

(1) When a device is in a low energy state, its remaining task queue needs to be rescheduled

(2) When a device detects that it cannot finish executing a task before the deadline of a task, it needs to reschedule the task

(3) When there is a change in the connection between devices that affects the migrated tasks, the tasks need to be rescheduled

Therefore, dynamic task scheduling is required for mobile devices when the above situations occur. In the following section, the important concepts and models in dynamic task scheduling are first described in detail.

4.1. Device-Cloud Connection Model. Since the mobile devices are in motion in the MDC environment, they are not in the same position. In this case, a certain device may disconnect from another device, microcloud, or central cloud at any time and, at the same time, establish a new connection with another device, microcloud, or central cloud, which leads to the fact that the task migration has to be adjusted at any time according to the change of the position between the devices and the connection to ensure that the

task can be completed successfully. And the result can also be returned normally.

We define the changes in the connection model between the device and the cloud at different moments during the execution of the task by the device due to the changes in the deployment location of the microcloud and the location of each mobile device.

As shown in Figure 6, we define four dynamic connectivity models between devices and the cloud and the corresponding changes in task migration due to the mobility of the device when the device hands over the task to another device or the cloud for execution.

4.1.1. Migration Model between Microclouds. In Figure 6(a), the initial situation is that devices A and B are in the network coverage of base station RSU-1 and therefore connectable to microcloud Cloudlet-1, devices D and E are in the network coverage of microcloud base station RSU-2 and therefore connectable to microcloud Cloudlet-2, and device A migrates a certain task to be executed by microcloud Cloudlet-1 in the base station in the area covered by RSU-1. When device A moves from the area covered by RSU-1 to the area covered by RSU-2, it means that device A will be disconnected from the microcloud Cloudlet-1 and enter the network coverage area of the microcloud Cloudlet-2, and therefore, the task that was originally performed by Cloudlet-1 needs to be transferred to be performed by Cloudlet-2, and then, device A will connect to Cloudlet-2 by connecting to RSU-2 to continue the communication.

4.1.2. Microcloud Internal Migration Model. In Figure 6(b), the initial scenario is that devices A and B are in the network coverage of base station RSU-1 and therefore connectable to the microcloud Cloudlet, devices D and E are in the network coverage of microcloud base station RSU-2 and therefore also connectable to the microcloud Cloudlet, and device A migrates a certain task to be executed by the microcloud Cloudlet in the base station RSU-1's region. When device A moves from the area covered by RSU-1 to the area covered by RSU-2, since both RSUs are connected to the same microcloud, it can continue to communicate with the microcloud Cloudlet even if there is a change in the network to which device A is connected, so there is no need to migrate the task, and it is sufficient that it is still left to be executed by the microcloud Cloudlet. However, switching the signal from RSU-1 to RSU-2 requires a time interval, so the connection between device A and the microcloud may be temporarily disconnected.

4.1.3. Microcloud-Device Intercloud Migration Model. In Figure 6(c), the initial situation is that devices A, D, and E are in the network coverage of the base station RSU-2 and therefore can be connected to the microcloud Cloudlet, device B is in the network coverage of the microcloud base station RSU-1 and therefore can also be connected to the microcloud Cloudlet, and at the same time, devices A, D, and E form a single cloud of devices and device E relocates the tasks to be executed by device A. When device A moves from the area covered by RSU-2 to the area covered by RSU-1, the connection between E and A is disconnected, so the device cloud formed by A, D, and E is dissolved, but devices D and E are still friendly neighbors and can still form a mobile device cloud. At this point, since device E has a task to give to device A for execution, the task needs to be given to another device to continue execution due to the movement of device A. At this time, since device D and device E are friendly neighbors, the task can be handed over to device D for execution, while at the same time, device E is in the area covered by RSU-2, i.e., device E can continue to communicate with the microcloud Cloudlet, and therefore, the task can also be handed over to the microcloud for continued execution.

4.1.4. Microcloud-Central Intercloud Migration Model. In Figure 6(d), the initial situation is that devices A and D are in the network coverage range of the base station RSU-2 and therefore can connect to the microcloud Cloudlet and device B is in the network coverage range of the microcloud base station RSU-1 and therefore can also connect to the microcloud Cloudlet, but device E is neither in the network coverage range of the RSU-1 nor in the network coverage range of the RSU-2, and therefore, device E is not connectable to the microcloud Cloudlet. Meanwhile, devices A and E form a device cloud, and E is not connectable to D. Device E migrates tasks to device A for execution. When device A moves from the area covered by RSU-2 to the area covered by RSU-1, the connection between E and A is disconnected, and at this time, since devices D and E are not connectable and E is not covered by RSU-2, device E cannot hand over the task to other devices or Cloudlet for execution, and then, device E can only choose to transfer the task to the central cloud for further execution. In this case, device E will connect to the central cloud via long-distance communication, such as 5G, and continue to execute the task.

As mentioned above, when there is a situation where a task needs to be rescheduled due to the movement of a device, when a device does not have enough energy to execute its remaining task queue, or when it is not able to finish executing a task before the deadline of the task, it is necessary to schedule the task immediately to ensure that it executes properly and returns the results.

4.2. Dynamic Scheduling System Model. The model is divided into three main parts, each of which has its functional module that handles a variety of information. The first part is the mobile device cloud, in which devices are friendly neighbors that can transfer tasks and return results to each other. When device A needs to migrate out a task, it immediately searches for friendly neighbors that can perform this task, and if found, it migrates this task to the most suitable mobile device B for execution according to the scheduling algorithm; the second component is the microcloud, which can receive task requests from device A and can communicate with all devices under the Wi-Fi coverage of this microcloud, scheduling to find a task-executable device C and migrate it to the mobile device as an intermediate point for transferring data between the original mobile device A and mobile device C. If a task-executable device is not found, the

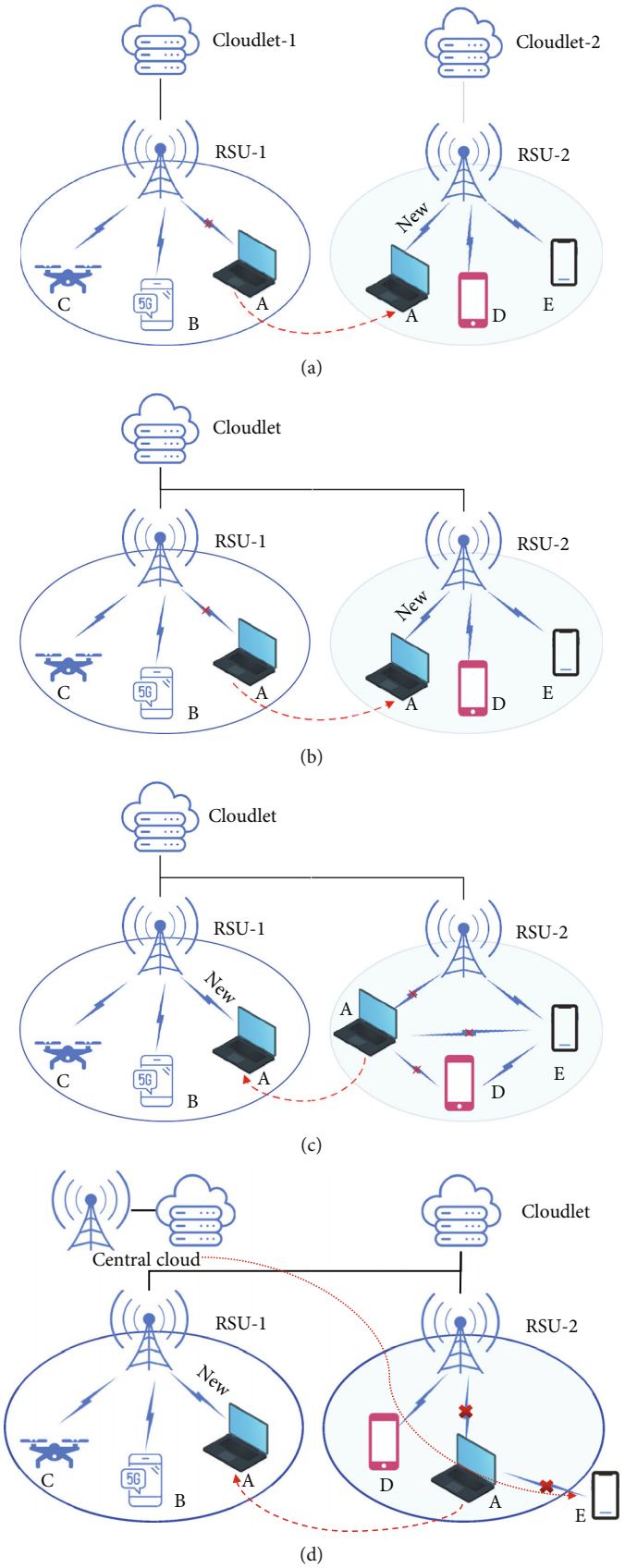


Figure 6: Mobile device-cloud connection model.

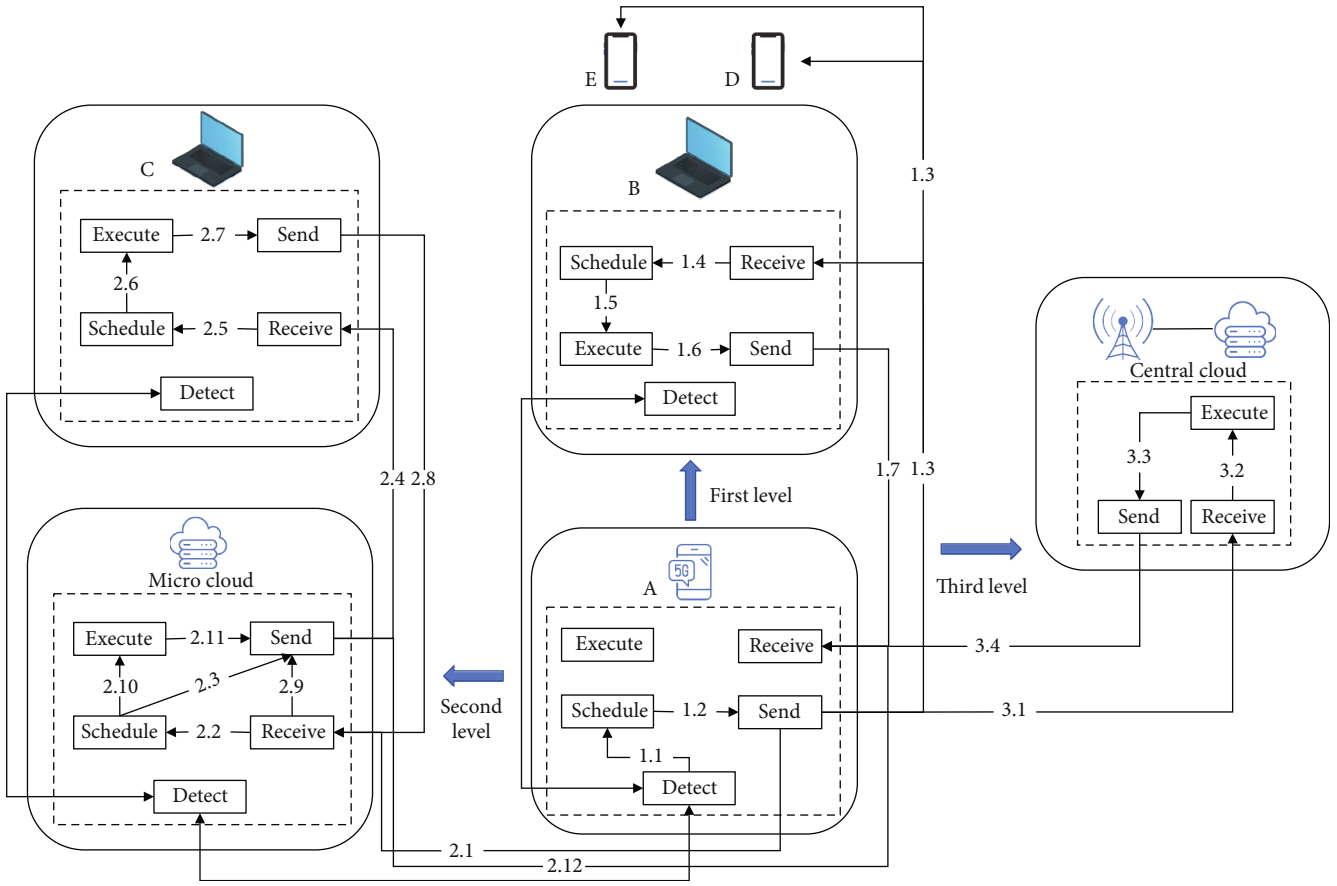


Figure 7: Overall model of the dynamic scheduling system.

microcloud can also give the task to be executed locally at an additional cost; the third component is the central cloud, which can be used to perform tasks that need to be migrated when neither the mobile device nor the microcloud can do the job, but this comes at an additional cost.

Figure 7 shows the overall model of the dynamic scheduling system.

The dynamic scheduling system mainly consists of the following components, each of which includes different modules:

(i) Mobile device function module: detect module, execute module, schedule module, and receive module

(ii) Microcloud functional modules: detect module, execute module, schedule module, and receive module

(iii) Central cloud functional module: detect module, execute module, and receive module

Each module is responsible for different information processing, and these functional modules interact with the functional modules of other devices to accomplish tasks together.

4.2.1. Detailed Working Model Description of the Dynamic Scheduling System. As shown in the dynamic scheduling system structure model diagram above, the specific working modes of the scheduling system are as follows.

(1) The First Level of Dynamic Scheduling. When the detection module of a device detects the need for scheduling tasks, it first needs to judge which task or tasks are most suitable for scheduling through the scheduling module and then send the handshake protocol to the friendly neighboring devices of the device through the sending module of the device. After receiving the handshake protocol, the friendly neighboring device will hand over the handshake protocol to the scheduling module for analysis and then return the handshake protocol through the sending module.

When the requesting device receives the returned handshake protocol, it makes the judgment on all the returned protocols, analyzes the friendly neighboring devices that can receive the task, selects the most suitable device, and sends out the task to be relocated through the sending module, and the receiving device receives the task through the receiving module and then hands it over to the scheduling module of the device for arranging the order of processing the task. After giving the task to the execution module for execution, the task result is returned to the requesting device through the sending module.

At the same time, the detection module of the requesting device and the receiving device will detect the movement

and position of each other, and if it is found that the connection with each other will be disconnected, it will notify each other in time, and then, the requesting device needs to restart the scheduling module and reschedule the uncompleted task to the other device to continue to complete.

(2) The Second Level of Dynamic Scheduling. When the requesting device receives the returned handshake protocol from a friendly neighboring device and judges that there is no friendly neighboring device that can accept the task that needs to be migrated, it sends the handshake protocol to the microcloud that can be connected through the sending module of the device. After the microcloud receives the handshake agreement, it hands over the handshake agreement to the scheduling module for analysis, and the scheduling module determines which mobile devices are within the network coverage area of the microcloud and then sends the handshake agreement to these mobile devices through the sending module of the microcloud.

When the mobile device receives the handshake protocol, it gives the handshake protocol to the scheduling module for analysis and then returns the handshake protocol through the sending module. When the microcloud receives the returned handshake protocol, it judges all the returned protocols, analyzes them for the devices that can receive the task, selects the most suitable device, and returns the handshake protocol to the requesting device through the sending module. The scheduling module of the requesting device selects the most suitable device by analyzing all the returned results and then sends the information to the receiving module of the microcloud through the sending module. The microcloud then sends out the tasks that need to be migrated through the sending module, the receiving device receives the tasks through the receiving module, then the scheduling module of the device arranges the order of the tasks to be processed, and finally, the task is handed over to the executing module to execute the tasks. At the same time, the detection module of the microcloud and the receiving device will detect each other's movement and position, and if it is found that the connection with each other will be disconnected, it will notify the other party in time, and then, the microcloud needs to restart the scheduling module and reschedule the unfinished tasks to other devices to continue to complete. After the task is completed, the device returns the result to the microcloud and then to the requesting device.

If the microcloud does not find a device that can accept the task after receiving the returned handshake protocols from all the devices within the network coverage, it will judge whether it can complete the task through the scheduling module and then return the handshake protocols to the receiving module of the requesting device through the sending module, and the scheduling module of the requesting device, after judgment, decides which microcloud is the most appropriate for executing the task and then sends the task to the receiving module of the microcloud through the sending module. After the microcloud accepts the task, it is scheduled by the scheduling module and then handed over to the execution module to complete the task. After completing the task, the result is returned to the requesting device through the sending module.

At the same time, the detection module of the requesting device and the microcloud will detect each other's movement and position, and if it finds that the connection with the other party is going to be disconnected, it will notify the other party promptly, and then, the requesting device will need to restart the scheduling module and then reschedule the uncompleted task to the other device to continue to complete.

(3) The Third Level of Dynamic Scheduling. If neither the friendly neighboring devices of the requesting device nor the microcloud returns the handshake protocol and finds a device that can accept the task, the task is sent to the receiving module of the central cloud through the sending module, and the central cloud accepts the task and gives it to the execution module to complete the task. After completing the task, the result is returned to the requesting device through the sending module.

At the same time, the detection module of the requesting device and the central cloud will detect each other's movement and position, and if it finds that the connection with each other will be disconnected, it will notify the other party in time, and then, the requesting device needs to restart the scheduling module and reschedule the uncompleted task to the other device to continue to complete.

4.3. Task Scheduling Decision-Making and Algorithm Design. To ensure optimal task scheduling, we propose a handshake protocol. This protocol involves a device sending handshake information to neighboring devices before scheduling a task. It aims to identify the most suitable neighboring device for task transfer. Upon receiving this information, neighboring devices assess their ability to execute the task and respond accordingly. This process helps the original device make informed scheduling decisions. In this way, if the friendly neighboring device cannot perform other redundant tasks, it can directly return a "no" message, so that the original device does not have to take this device into account when scheduling. When the friendly neighboring device can complete the task, it will return other necessary reference information, and then, the original device can make a judgment based on this information, which can improve the scheduling efficiency.

Figure 8 shows the process of scheduling at the first level, i.e., within the mobile device cloud, when the requesting device needs to schedule a task.

5. Experiments and Analysis of Results

5.1. Experimental Program

5.1.1. Experimental Environment. The system development in this paper is divided into two main parts: task

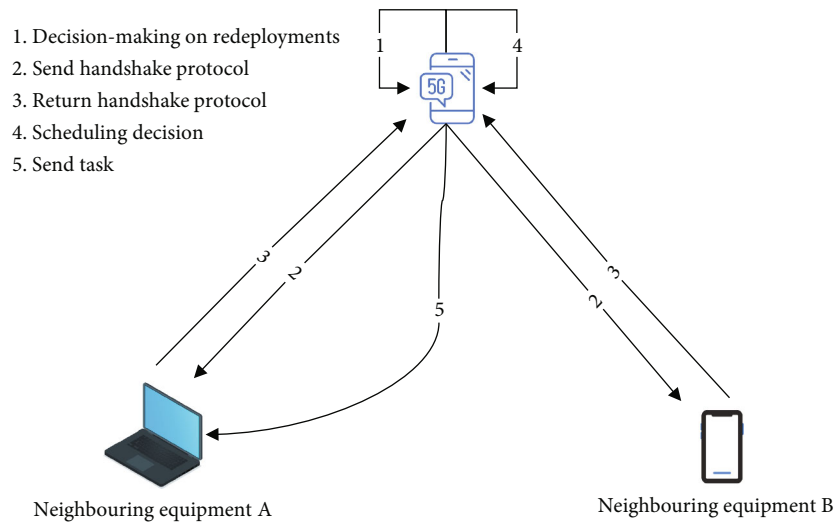


Figure 8: Schematic diagram of task scheduling within the mobile cloud.

decomposition and simulation testing of the scheduling system. The experimental part of this paper is developed in C++ programming language, under the Windows 10 system, using Visual Studio 2021. The experimental environment is as follows:

(1) Operating system: Windows 10

(2) Memory size: 32 G

(3) CPU: Intel® Core™ 2 Duo CPU 2.00 GHz

(4) Development platform: Visual Studio 2021

5.1.2. Experimental Program on Task Decomposition. In this paper, the decomposition process of tasks is completed by analyzing computationally intensive task procedures, transforming tasks into graph structures, then merging the nodes in the graph into independent nodes, and finally merging the independent nodes into business logic units. Among them, for the generation of task graph structure, the test procedures in this paper are classical computationally intensive procedures, including matrix multiplication procedure, matrix inverse procedure, Gaussian function procedure, convex package problem procedure, and traveling quotient problem procedure.

The entire procedure is transformed into a graph structure, where each node holds detailed information related to that node for subsequent analysis. The data structure of the specific task structure graph is defined as Algorithm 2.

For different types of statements and structures in a program, the corresponding data structures and methods of constructing nodes are different. Based on the task structure graph, the graph structure is retraced from bottom to top, and if the data dependency value between a node and the previous node is greater than the threshold set in this paper, the two nodes are merged into an independent unit. Finally, the independent units are merged based on business logic to get the final subtask.

5.1.3. Experimental Program on Task Scheduling Algorithms. Based on the previously described scheme, an experimental comparison of the performance of the scheduling algorithm of this paper and the random scheduling algorithm is carried out. The experimental data generation scheme is as follows:

(1) Randomly generate the matrix ETC$[m, n]$, where each element size is within the range of Fan $[1, 100]$, and ETC$[i, j]$ denotes the estimated running time of subtask T_i on mobile device P_j

(2) Randomly generate the matrix Trans$[m, n]$, where each element size is in the range of $[1, 100]$, and Trans$[i, j]$ denotes the data transmission delay between device P_i and device P_j

(3) A subtask structure graph is a directed loop-free graph representing scheduling constraint relationships between individual subtasks, which is generated by the scheme described above. For a structure graph G, let $G = <T, E, \text{ETC}>$, where T is the set of subtasks, which is the set of directed edges in the E subtask structure graph, and ETC is the estimated running time matrix

Based on the above data, the verification of the experimental performance of the task scheduling algorithm is carried out. Among them, the number of mobile devices is set to five, their initial coordinate positions are randomly generated, and the horizontal and vertical coordinate ranges are all within the range of $[1, 500]$, and there are two microclouds with randomly generated coordinates. The experimental program is that the tasks are scheduled with the algorithm of this paper and the random scheduling algorithm, respectively, and for each group of tasks, the test is performed ten times, and the average of the time-consuming and energy-consuming values is recorded. Among them, the stochastic algorithm uses random assignment of tasks in the initial assignment of tasks, and in the

```
class Graph{
string name;          //name of the function
int anum;             //number of arguments to the function
vector<string> argname;                    //list of arguments of the function
int begin_pos;                 //start position of the node
int end_pos;                  //end position of the node
bool isok;               //Is the function built?
string type;                  //The type of the node
int relation;                    //To determine the relation of the node to the previous node when traversing from the bottom up.
string content;                    //Content of the node
Graph * next;                    //Points to the next node which is the node of the iterative relation.
Graph * brother;                          //Points to the brother node, the node in the juxtaposition relationship.
Graph * father;                 //Points to the father node.
Graph * child;                  //Points to the child node.
};
```

Algorithm 2: Definition of data structures for task structure graphs.

subsequent dynamic scheduling process, if a device needs to schedule a task, a task is randomly selected to be called out and selected to be randomly dispatched to a device for execution.

5.2. Analysis of Experimental Results

5.2.1. Generation of Task Structure Diagrams. To decompose the task into subtasks, it is first necessary to construct a task structure diagram based on the task procedure before the next step of analysis can be carried out. The process of constructing a task structure diagram involves analyzing the statements and structure of the program using the features of sequential, branching, and looping structures and then constructing the structure diagram. Figure 9 shows a sample program and the corresponding constructed task structure diagram.

The part of the task structure diagram that builds the result of running the program is shown in Figure 10.

Once the task structure graph is constructed, the nodes in the graph are traversed from the bottom up, and independent units are constructed by scanning and tracking large data and merging neighboring nodes whose data dependencies exceed a specific threshold. And then through the concept of the business logic unit, the independent units are merged to construct the business logic unit, and finally, the task is decomposed into a series of subtasks. Figure 10 shows the running screenshot of the construction procedure of the task structure graph.

5.2.2. Impact of Combining Standalone Units on Energy Savings. Due to the findings in the literature [22] that the mobile device migration task activates the network interface of the device when the task is successfully transmitted, the interface does not immediately switch to a low-power state but continues to remain in a high-power state for tens of seconds, which leads to unnecessary energy loss; to address this issue, this paper adopts the scheme of merging independent units into business logic units to reduce unnecessary energy loss.

According to the literature [22], for the 5G interface of iOS9, the tailing time is measured to be 9 seconds. Therefore, based on this theory, the scheme of merging independent units is experimentally tested in this paper. The specific scheme is to randomly schedule tasks between devices in the case of merging standalone units into business logic units and in the case of not merging standalone units, respectively, and, at the same time, record the total energy consumption of all devices. In this case, the total number of devices is 5, and the number of tasks varies incrementally.

Figure 11 shows a comparison of the energy consumption of mobile device migration tasks based on standalone units and business logic units, respectively. The number of tasks refers to the number of independent units, and the number of tasks varies from 10 to 80, and the energy consumption required for the execution of each task on the device is randomly generated, with case 1 being "subtasks based on independent units" and case 2 being "subtasks based on business logic units." In case 1, 60% of the tasks are randomly selected for scheduling, and in case 2, 60% of the tasks are still selected for scheduling, trying to make the basic known content consistent in both cases. As can be seen from the figure, the energy consumption of the equipment is significantly reduced after merging the standalone units, and, as the number of tasks increases, the energy consumption advantage becomes more obvious. Therefore, merging standalone units into business logic units extends the life cycle of the device. The business logic unit is the smallest logical unit that the user can feel, so compared to the standalone unit, migrating the tasks of the business logic unit between devices is completely acceptable to the user, even though the latency will increase.

5.2.3. Effect of Parameters of Initial Task Scheduling Algorithm Based on Genetic Algorithm. The initial task scheduling algorithm in this paper is designed based on the genetic algorithm, and in the genetic algorithm, several important parameters include crossover rate, mutation rate, maximum number of iterations, and initial population size, which need to be considered to be determined. At present, researchers generally believe that a crossover rate of 0.6 to

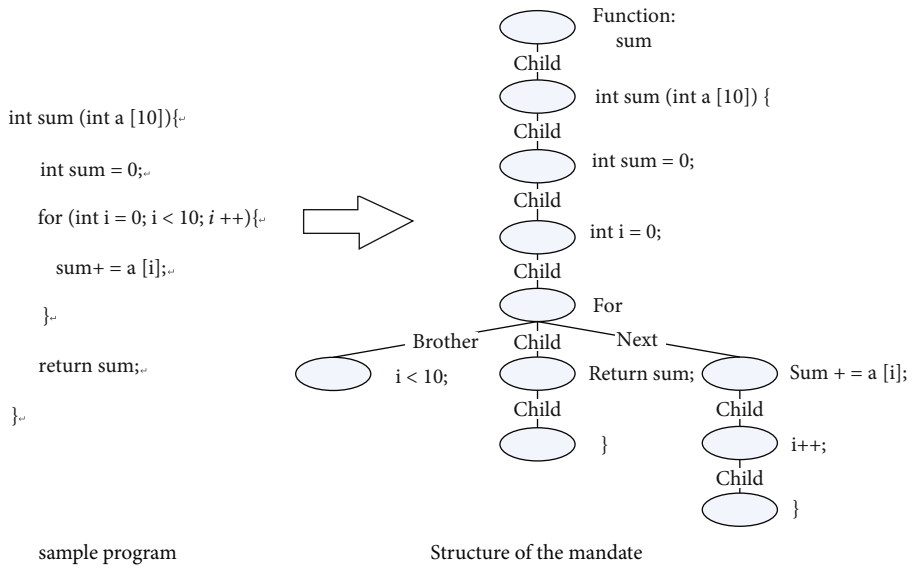


Figure 9: Task structure diagram schematic.

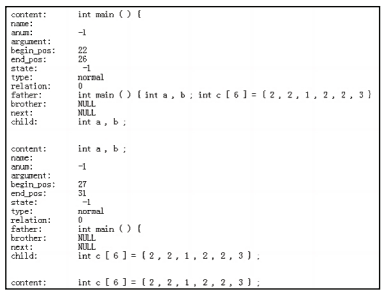


Figure 10: Screenshot of the constructor running for the task structure diagram.

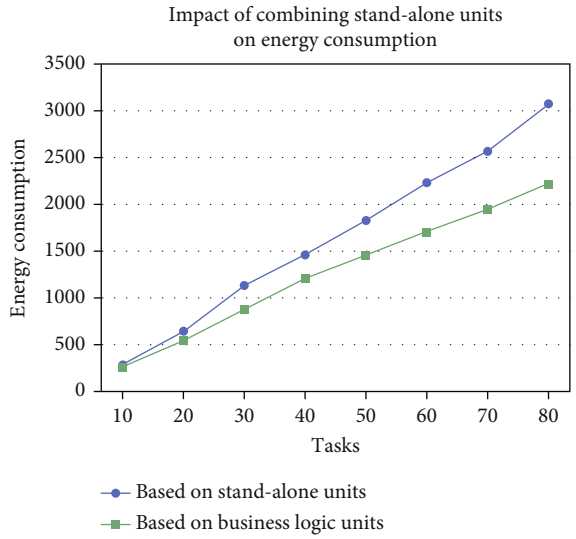


Figure 11: Based on standalone units vs. business logic units.

Table 1: Testing the parameters of the initial task scheduling algorithm based on the genetic algorithm.

No.	PXOVER	PMUTATION	Time (s)	Energy (J)
1	0.8	0.1	371	540
2	0.8	0.2	418	552
3	0.8	0.3	461	596
4	0.8	0.4	483	733
5	0.7	0.1	323	668
6	0.7	0.2	341	661
7	0.7	0.3	351	651
8	0.7	0.4	412	597
9	0.6	0.1	358	565
10	0.6	0.2	333	580
11	0.6	0.3	346	600
12	0.6	0.4	427	635

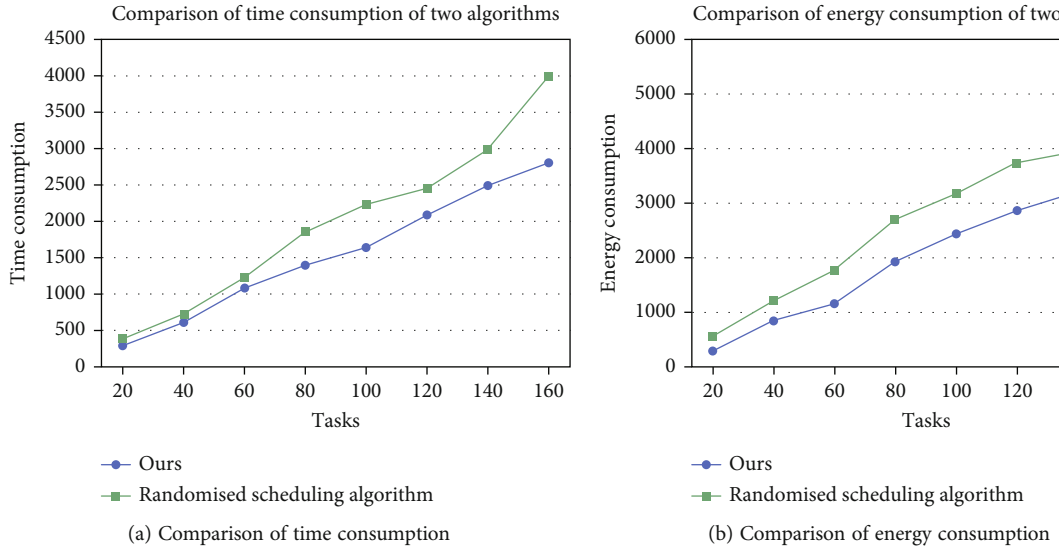
Table 2: Determination of initial scheduling algorithm parameters.

No.	Parameters	Value
1	Crossover rate	0.8
2	Variation rate	0.1
3	Population size	100
4	Maximum number of iterations	1000

1 and a variation rate of about 0.1 to 0.4 will be more favorable to the results of the genetic algorithm. Therefore, in this section of the paper on scheduling, the above two parameters are first tested to find the best value for this problem. As shown in Table 1, for a given known background, i.e., 5 mobile devices, 50 tasks, and other conditions are also determined, the values of time to task completion and total

Table 3: Performance comparison between this algorithm and randomized algorithm.

No.	Task_num	Time_GA (s)	Time_random (s)	Energy_GA (J)	Energy_random (J)
1	10	121	235	198	258
2	20	291	392	296	564
3	30	463	623	601	831
4	40	606	724	847	1208
5	50	818	1011	999	1336
6	60	1082	1227	1157	1766
7	70	1312	1558	1622	2087
8	80	1393	1851	1921	2696



(a) Comparison of time consumption

(b) Comparison of energy consumption

Figure 12: Comparison of the performance of this algorithm and the randomized algorithm.

energy consumption change when different combinations of crossover rate (PXOVER) and mutation rate (PMUTATION) are taken. In this, for each case, 10 sets of data are tested and then averaged.

As can be seen from the table, when PXOVER is 0.8 and PMUTATION is 0.1, the energy and time consumption are relatively small, so we define the relevant parameter values as shown in Table 2.

(1) The crossover rate determines the length of the chromosome segments exchanged during chromosome crossing over and is taken as 0.8

(2) The mutation rate determines the number of genes that are mutated during chromosome mutation and is taken as 0.1

(3) Define the population size as 100

(4) The maximum number of iterations is defined as 1000

5.2.4. Evaluation of Scheduling Algorithm Performance. Based on the previously described scheme, we conducted an experimental comparison of the performance of the scheduling algorithm of this paper and the random scheduling algorithm, assuming five devices, respectively, and the predicted running time of each subtask on different mobile devices and the predicted transmission delays between the mobile devices are randomly generated. The tasks were scheduled using the algorithm of this paper and the random scheduling algorithm, respectively, and for each set of tasks, the tests were performed ten times, and the average of the elapsed time and energy consumption values was recorded. Among them, the random algorithm uses random assignment of tasks in the initial assignment of tasks, and in the subsequent dynamic scheduling process, if a certain device needs to schedule a task, a task is randomly selected to be called out and selected to be randomly scheduled to a certain device for execution.

Table 3 shows the performance comparison between this paper's algorithm and the randomized algorithm. As shown in Figure 12 for the test results of the two algorithms, it can be seen that the scheduling algorithm based on the genetic algorithm in this paper is better than the random scheduling algorithm in terms of energy consumption and throughput rate, respectively, and in terms of time consumed, the

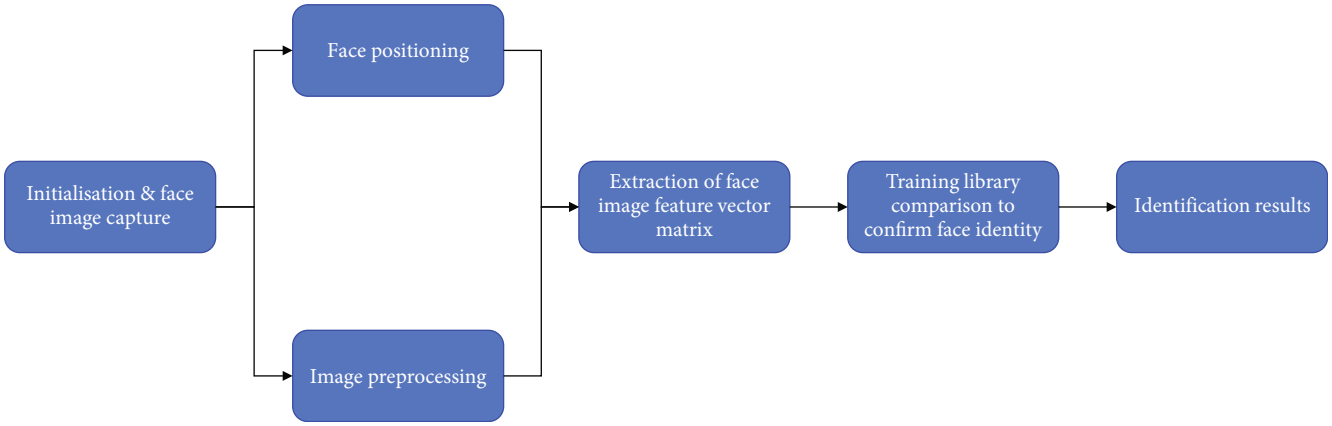


Figure 13: Data flow model for face recognition application computational node.

advantage increases significantly with the increase in the number of tasks. The reason is mainly that the genetic algorithm is an optimization scheme formed after exploring the whole domain of the problem space, so it has better adaptability to different task loads and task types, and the algorithm has higher robustness and efficiency.

From the above figure, it can be seen that the algorithm of this paper is better than the random scheduling algorithm; this is because, in the process of initial task allocation, this paper adopts the genetic algorithm for scheduling. As discussed in Section 3.2, the genetic algorithm finds the most robust solution by comprehensively exploring the problem space. This approach maximizes the fitness function (equation (2)) $F(x)$, ensuring the minimization of the objective function (equation (1)) $f(x)$. Given the known information required for task scheduling, the genetic algorithm can find a better solution in a relatively short period, and the time consumed is relatively short. In the subsequent dynamic scheduling process, since the algorithm adopted in this paper first performs the call-out decision judgment within the requesting device and selects the optimal device among the neighboring devices for scheduling, the optimal solution can be obtained, which saves energy consumption and reduces the time delay. In conclusion, the proposed task scheduling algorithm for mobile device cloud has better performance; in particular when the number of tasks increases, it can save a lot of energy consumption and time.

5.3. Practical Application Environment Validation. Face recognition, i.e., an application that acquires a stream of image data from a camera in real time, is currently used in several domains. We have applied the computational slicing method of the stateful data stream application studied in this paper to the face recognition program for Android mobile of our related work and built a backend server for face recognition image processing and training library. The computational nodes of this face recognition program can be divided into initialization and face image ingestion nodes, face localization nodes, image preprocessing nodes, extraction of face image feature vector matrix nodes, training library comparison to confirm face identity nodes, and face identity recognition result confirmation nodes. The data flow model of the relationship of each computational node is shown in Figure 13.

Table 4: Application detail parameters.

(a)

Node number	Local execution time	Execution time on cloud	State data volume
①	1.5 s	—	2.3 MB
②	0.9 s	0.093 s	1.2 MB
③	0.7 s	0.065 s	3.1 MB
④	1.1 s	0.012 s	1.1 MB
⑤	2.6 s	0.023 s	2.4 MB
⑥	0.3 s	—	0.1 MB

(b)

Transmission side	Transmission data volume
①-②	2.3 MB
①-③	2.3 MB
②-④	1.2 MB
③-④	1.5 MB
④-⑤	0.5 MB
⑤-⑥	0.1 MB

Initialization and face image ingestion are the first nodes that must be performed on the mobile, and the identity confirmation result is the last node, which is ultimately performed on the mobile. Therefore, while the mobile application implements the above computation nodes, the server also implements the face positioning computation interface, the image preprocessing interface, the interface for extracting the feature vector matrix of the face image, and the interface for confirming the identity of the face by comparing the training libraries. The detailed parameters of the application are given in Table 4.

The initial computational cut-schemes in the application are all locally executed, i.e., initial chromosome {1, 1, 1, 1, 1,

Table 5: Record of experiments.

Bandwidth	Task scheduling solution	Execution time	All nodes local	All nodes cloud
832.1 KB/s	1, 1, 1, 0, 1, 1	5.69 s	6.02 s	5.71 s
1178.8 KB/s	1, 1, 1, 1, 0, 1	5.21 s	6.15 s	5.29 s
1406.3 KB/s	1, 1, 0, 1, 1, 1	4.33 s	5.98 s	4.93 s
1561.2 KB/s	1, 1, 0, 1, 0, 1	4.01 s	6.13 s	4.67 s
1684.3 KB/s	1, 1, 0, 1, 0, 1	3.98 s	6.36 s	4.69 s
1750.4 KB/s	1, 1, 0, 0, 0, 1	3.41 s	6.24 s	4.33 s
1928.7 KB/s	1, 0, 1, 0, 0, 1	3.42 s	5.89 s	4.12 s
1954.8 KB/s	1, 1, 0, 0, 0, 1	3.01 s	6.11 s	3.98 s
2249.4 KB/s	1, 1, 1, 0, 0, 1	2.82 s	5.21 s	3.81 s
2288.1 KB/s	1, 1, 1, 0, 0, 1	2.71 s	6.22 s	3.92 s

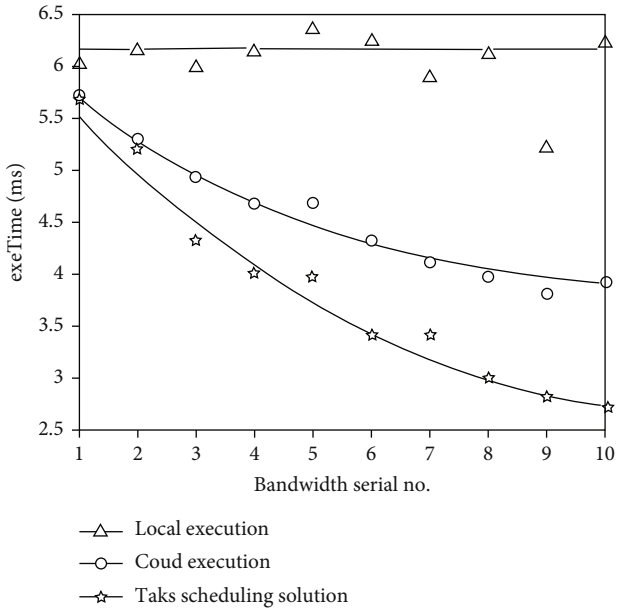


Figure 14: Recorded results of the experiment.

1}, and log logging is done to record the network bandwidth, node cut-schemes, and execution time. We did 50 sets of experiments, and Table 5 shows the top 10 sets of records sorted by network bandwidth.

As shown in Figure 14, from the analysis of the experimentally recorded results, the computational slicing approach performs better as the network bandwidth increases and the application execution time decreases. The scheme with all nodes executing on the cloud also decreases its execution time as the network bandwidth increases, but the slope of the decrease is smaller than that of the computational slicing scheme. The scheme uses all nodes executing locally; basically, its execution time is larger and has the worst performance.

6. Conclusion

In this paper, we proposed a task decomposition strategy and task scheduling algorithm for mobile devices in a cloud environment for the task scheduling problem to maximize the data throughput rate of mobile devices by minimizing energy consumption. The main research content of this paper is as follows.

(1) Combined with the characteristics of the mobile device cloud, this paper constructed a dynamic task scheduling model by taking advantage of the overall exploration strategy of the genetic algorithm and the optimization search method, which did not rely on the gradient information or other auxiliary knowledge in the computation process, and only relied on the characteristics of the objective function and the corresponding fitness function that affects the search direction. Aiming at the objectives of minimizing energy consumption and maximizing throughput rate, task scheduling algorithms in the MDC environment are proposed, including the task scheduling algorithm based on a genetic algorithm, which was applied to the allocation process of subtasks at each level

(2) The correctness of the granularity of this task decomposition algorithm was verified by experimentally proving that merging independent units could reduce energy consumption, and the parameters of the genetic algorithm-based task scheduling algorithm were tested to determine the optimal parameters in the MDC environment

(3) The task scheduling model based on the genetic algorithm and random scheduling algorithm were compared by comparison experiments, which showed that the allocation time of the task scheduling model based on the genetic algorithm was shortened by 11.82%~48.51% and the energy consumption was reduced by 22.28%~47.52% under different load conditions

The rapid development of IoT and the continuous emergence of new types of sensors and mobile terminals bring numerous challenges to MDC task scheduling. In future research, the research on data privacy protection of task scheduling algorithms in the MDC environment should be strengthened to prevent the leakage of users' private data.

Data Availability

The dataset used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] Y. Hua, B. Xiao, and X. Liu, "Nest: locality-aware approximate query service for cloud computing," in *2013 Proceedings IEEE INFOCOM*, pp. 1303–1311, Turin, Italy, 2013.

[2] J. Chen, T. Li, Y. Zhang et al., "Global-and-local attention-based reinforcement learning for cooperative behaviour control of multiple UAVs," in *IEEE Transactions on Vehicular Technology*, IEEE, 2023.

[3] Y. Li and J. Cao, "Adaptive binary particle swarm optimization for WSN node optimal deployment algorithm," *IECE Transactions on Internet of Things*, vol. 1, no. 1, pp. 1–8, 2023.

[4] J. Paczkowski, "iPhone owners would like to replace battery," *All Things Digital*, vol. 21, 2009.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[6] C. Liu, Y. Yao, Y. Dang et al., "WMDRS: workload-aware performance model based multi-task dynamic-quota real-time scheduling for neural processing units," in *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*, Nanjing, China, 2023.

[7] Z. Zabihi, A. M. Eftekhari Moghadam, and M. H. Rezvani, "Reinforcement learning methods for computation offloading: a systematic review," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–41, 2024.

[8] S. W. Loke, "The internet of flying-things: opportunities and challenges with airborne fog computing and mobile cloud in the clouds," 2015, https://arxiv.org/abs/1507.04492.

[9] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6898–6908, 2020.

[10] Z. Xie, X. Song, J. Cao, and W. Qiu, "Providing aerial MEC service in areas without infrastructure: a tethered-UAV-based energy-efficient task scheduling framework," *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 25223–25236, 2022.

[11] J. Zhou, T. Wang, W. Jiang, H. Chai, and Z. Wu, "Decomposed task scheduling for security-critical mobile cyber–physical systems," *IEEE Internet of Things Journal*, vol. 9, no. 22, pp. 22280–22290, 2022.

[12] J. Chen, P. Han, Y. Zhang, T. You, and P. Zheng, "Scheduling energy consumption-constrained workflows in heterogeneous multi-processor embedded systems," *Journal of Systems Architecture*, vol. 142, p. 102938, 2023.

[13] J. Sun, Y. Zhang, Z. Wu et al., "An efficient and scalable framework for processing remotely sensed big data in cloud computing environments," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4294–4308, 2019.

[14] C. Zhou, W. Wu, H. He et al., "Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 911–925, 2021.

[15] Y. Kim, C. Song, H. Han, H. Jung, and S. Kang, "Collaborative task scheduling for IoT-assisted edge computing," *IEEE Access*, vol. 8, pp. 216593–216606, 2020.

[16] M. K. Pandit, R. N. Mir, and M. A. Chishti, "Adaptive task scheduling in IoT using reinforcement learning," *International Journal of Intelligent Computing and Cybernetics*, vol. 13, no. 3, pp. 261–282, 2020.

[17] J. Yang, B. Jiang, Z. Lv, and K. K. R. Choo, "A task scheduling algorithm considering game theory designed for energy management in cloud computing," *Future Generation Computer Systems*, vol. 105, pp. 985–992, 2020.

[18] A. Lakhan, J. Li, T. M. Groenli et al., "Dynamic application partitioning and task-scheduling secure schemes for biosensor healthcare workload in mobile edge cloud," *Electronics*, vol. 10, no. 22, p. 2797, 2021.

[19] B. Du, N. Shan, and S. Zhou, "An efficient Mobile cloud service model or tactical edge," in *In Advances in Intelligent Networking and Collaborative Systems: The 11th International Conference on Intelligent Networking and Collaborative Systems (INCoS-2019)*, pp. 422–430, Springer International Publishing, 2020.

[20] K. Yang, S. Ou, and H. H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *IEEE Communications Magazine*, vol. 46, no. 1, pp. 56–63, 2008.

[21] V. Jafari and M. H. Rezvani, "Joint optimization of energy consumption and time delay in IoT-fog-cloud computing environments using NSGA-II metaheuristic algorithm," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 3, pp. 1675–1698, 2021.

[22] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, set, go: coalesced offloading from mobile devices to the cloud," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, Toronto, ON, Canada, 2014.