Wuhan University
Journal of Natural Sciences

2026, Vol.31 No.1, 069-078

Article ID 1007-1202(2026)01-0069-10 DOI https://doi.org/10.1051/wujns/2026311069

Cite this article: LI Shuo, FANG Zuying, ZHOU Guoqiang, *et al*. A Real-Time Task Scheduling Algorithm Based on Bilateral Matching Games in a Distributed Computing Environment[J]. *Wuhan Univ J of Nat Sci*, 2026, 31(1): 69-78.

A Real-Time Task Scheduling Algorithm Based on Bilateral Matching Games in a Distributed Computing Environment

□ **LI Shuo[1], FANG Zuying[2], ZHOU Guoqiang[2], DAI Guilan[3†]**

1. School of Intelligent Manufacturing, Huanghuai University, Zhumadian 463000, Henan, China;
2. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, Jiangsu, China;
3. Department of Computer Science and Technology, Tsinghua University, Beijing100084, China

Abstract: In the era of the Internet of Things, distributed computing alleviates the problem of insufficient terminal computing power by integrating idle resources of heterogeneous devices. However, the imbalance between task execution delay and node energy consumption, and the scheduling and adaptation challenges brought about by device heterogeneity, urgently need to be addressed. To tackle this problem, this paper constructs a multi-objective real-time task scheduling model that considers task real-time performance, execution delay, system energy consumption, and node interests. The model aims to minimize the delay upper bound and total energy consumption while maximizing system satisfaction. A real-time task scheduling algorithm based on bilateral matching game is proposed. By designing a bidirectional preference mechanism between tasks and computing nodes, combined with a multi-round stable matching strategy, accurate matching between tasks and nodes is achieved. Simulation results show that compared with the baseline scheme, the proposed algorithm significantly reduces the total execution cost, effectively balances the task execution delay and the energy consumption of compute nodes, and takes into account the interests of each network compute node.

Key words: dispersed computing; real-time task; task scheduling; bilateral matching game

CLC number: TP301

0 Introduction

With the advent of the Internet of Everything (IoE) era, a dramatic increase in the number of Internet of Things (IoT) devices has been witnessed. These devices generally operate independently and exhibit a wide geographical distribution. Such a dispersed nature gives rise to a significant challenge, wherein the potential computational capability of numerous IoT devices is not fully utilized. To address this challenge, dispersed computing has emerged as a promising paradigm. Dispersed Computing aims to construct a decentralized, distributed, and resource-heterogeneous network entity by leveraging the computational resources of ubiquitous smart devices,

Received date: 2025-09-04

Foundation item: Supported by the National Program on Key Basic Research Project (2020YFA0713600) and the National Natural Science Foundation of China (62272214)

Biography: LI Shuo, female, Master candidate, research direction: computer applications, knowledge graph. E-mail: lishuo@huanghuai.edu.cn

† Corresponding author. E-mail: daigl@tsinghua.edu.cn

such as smartphones, tablets, and IoT terminals[1-2]. Its objective is to enhance the efficiency, security, and reliability of distributed systems without increasing centralized control, thereby compensating for the high latency and cost deficiencies of the cloud computing paradigm, and addressing the issues of limited coverage areas of edge nodes and the inability of terminal nodes to collaborate directly in mobile edge computing.

While dispersed computing improves system throughput by scheduling compute-intensive tasks to nearby available networked computation nodes for execution, performing rational scheduling for real-time tasks among these dispersed and heterogeneous devices poses significant challenges. First, the computational resources of networked computation nodes are limited, and they should be entitled to accept or reject task execution requests to guarantee their own operational stability. Second, significant differences exist among tasks in terms of data volume, computational workload, and completion deadlines; thus, task scheduling and resource allocation should be compatible with the interests of the tasks. Finally, while ensuring the real-time execution of tasks, the energy consumption of each networked computing node must also be considered, necessitating a balance between task execution latency and the energy consumption of these nodes.

To address the aforementioned challenges and resolve the imbalance in the trade-off between task execution latency and energy consumption in dispersed computing environments, this paper constructs a real-time task scheduling model for such environments. This model accommodates the interests of both tasks and networked computation nodes, aiming to minimize task execution latency and system energy consumption while maximizing overall system satisfaction, all under the constraint of ensuring real-time task execution. Furthermore, a real-time task scheduling algorithm based on a two-sided matching game is designed to obtain the scheduling strategy. This approach allows tasks and networked computation nodes to make selections proactively based on their preference rules, thereby achieving an optimal balance between task execution latency and the energy consumption of the nodes. Finally, through comparative analysis in simulation experiments from various perspectives, the effectiveness and superiority of the proposed algorithm are verified in terms of task execution latency, system energy consumption, satisfaction, and throughput.

The main contributions of this paper are summarized as follows:

1) A real-time task scheduling framework for dispersed computing environments is constructed, which is tailored to the characteristics of dispersed computing scenarios. This framework accommodates the interests of both tasks and networked computation nodes, with the optimization objective of minimizing task execution latency and system energy consumption while maximizing overall system satisfaction.

2) A real-time task scheduling algorithm based on a two-sided matching game is proposed and designed. Through the rational design of preference rules, a balance between task latency and the energy consumption of networked computation nodes is achieved, enabling an effective matching between tasks and networked computation nodes.

In this paper, extensive simulation experiments are designed to evaluate the performance of the proposed algorithm and the advantages of dispersed computing. The simulation results demonstrate that the algorithm proposed in this paper effectively balances the execution latency of tasks with the energy consumption of computing nodes, while accommodating the interests of all networked computation nodes.

1 Related Work

In the research area of dispersed computing, progress has been made on issues such as user anonymity processing[3], fine-grained data packet monitoring in networks[4], application-level congestion control[5], and privacy and security protection[6]. In terms of architectural design, Ghosh *et al*[7] proposed a container orchestration architecture for dispersed computing, while Yang *et al*[8] introduced a functional architecture for dispersed computing along with three core technologies.

The resource optimization approach of mobile edge computing provides an early technical reference for distributed computing. This type of research focuses on the joint optimization of task offloading and resource allocation. The core goal is to balance latency and energy consumption in a static network environment. An *et al*[9] decomposed the task offloading strategy, communication resource allocation and computing resource scheduling into multi-objective optimization problems for IoT edge scenarios. They solved the problem by using the golden

search method and achieved the minimization of energy consumption under latency constraints. Li *et al*[10] introduced potential game theory and designed a distributed task offloading mechanism. They used the game equilibrium characteristics to ensure the stability of energy consumption reduction. Song *et al*[11] further proposed an offloading algorithm based on network topology for multi-service scenarios. They reduced the overall system cost by refining the task type and node matching rules. However, such methods are essentially an extension of the static architecture of MEC. Their optimization premise is that the network topology and node resource status are relatively fixed. They do not consider the characteristics of dynamic node addition/exit and intermittent link interruption in distributed computing. For example, when faced with sudden traffic spikes or node failures, the existing offloading strategy cannot quickly adjust the task allocation logic, resulting in the failure of global real-time collaboration among multiple tasks, making it difficult to directly migrate to dynamic scenarios of distributed computing.

To adapt to the dynamic changes in the network environment, some studies have turned to reinforcement learning technology to achieve real-time optimization of scheduling strategies through the interaction between agents and the environment. Kuang *et al*[12] proposed an integrated framework based on deep reinforcement learning, which models task offloading, scheduling decisions and resource allocation as Markov decision processes and uses deep neural networks to fit the optimal strategy, effectively reducing system latency and energy consumption in multi-user scenarios. Sacco *et al*[13] further constructed a multi-agent reinforcement learning architecture, allowing each node to participate in decision-making as an independent agent, which improved the distributed collaboration capability of task scheduling in unmanned aerial vehicle (UAV) networks. Seid *et al*[14] extended this idea to multi-UAV IoT edge networks, and solved task offloading conflicts between heterogeneous nodes through multi-agent deep reinforcement learning. Although such methods are superior to traditional strategies in terms of dynamic adaptation, their limitations are still quite prominent: First, they rely on fixed infrastructure. When nodes in distributed computing are randomly distributed personal devices, infrastructure dependence leads to a significant decrease in the scalability of the algorithm. Second, they ignore the core advantage of mining the value of idle resources in distributed computing. That is, existing models mostly assume that node resources are pre-allocated dedicated resources, do not consider the dynamic utilization of idle computing power of neighboring nodes, and do not quantify the utility feedback of node contribution computing power, resulting in low resource utilization.

In response to the adaptability defects of the first two types of methods, recent research has begun to focus on the scenario characteristics of distributed computing and design dedicated scheduling frameworks. The core objective is to solve the problems of throughput optimization and extreme scenario adaptation in dynamic heterogeneous networks. Niu *et al*[15] constructed a distributed computing network architecture for disaster emergency scenarios and used Lyapunov optimization technology to handle the dynamics of task queues. While ensuring queue stability, they reduced system energy consumption and provided a feasible solution for distributed scheduling in extreme environments. Poylisher *et al*[16] designed an anti-interference scheduling framework for tactical edge networks and used fault tolerance mechanisms to deal with challenges such as node loss and bandwidth limitation, ensuring the continuous execution of tactical tasks. Yang *et al*[17] and Hu *et al*[18] made breakthroughs from the perspective of task models. The former used virtual queuing networks and maximum weight scheduling to achieve the optimal throughput of task chains and directed acyclic graph (DAG) models. The latter designed a throughput optimization scheduler for continuous and stable data input scenarios, which can still maintain efficient task processing when link bandwidth fluctuates and computational load is high. While such frameworks align with the distributed and dynamic requirements of distributed computing, significant research gaps remain. Existing work primarily focuses on increasing system throughput and shortening task execution time, with attention to node energy consumption limited to basic constraints such as not exceeding thresholds. A triangular balance mechanism between latency, energy consumption, and real-time performance has not been established. However, in distributed computing, many nodes are battery-powered mobile devices with limited energy resources. Simply pursuing throughput or low latency can lead to node energy overload, ultimately reducing long-term system stability—this core contradiction is a critical issue that current research has

yet to address.

In summary, while existing distributed computing scheduling research has made progress in dynamic adaptation and scenario-specificity, it still falls short in multi-objective collaborative optimization and the utilization of idle resources. This paper addresses this gap by constructing a scheduling model that balances task real-time performance, node energy consumption, and system satisfaction. Furthermore, it aligns the interests of tasks and nodes through a two-sided matching game, thus overcoming the limitations of existing research.

2 A Real-Time Task Scheduling Framework

Based on the presence or absence of task execution requests, networked computation nodes are categorized into two types: those with task execution demands are referred to as Task Nodes, while the remaining nodes with idle computational resources are termed Computation Nodes. It is assumed that within a given time interval, there exist M Task Nodes and N Computation Nodes, and the positions of these nodes do not change within this single interval. The M Task Nodes are denoted by the set $U \triangleq \{u_1, u_2, \cdots, u_i, \cdots, u_M\}$, where u_i represents the index of a Task Node, and each Task Node has several compute-intensive real-time tasks. The N Computation Nodes are denoted by the set $V \triangleq \{v_1, v_2, \cdots, v_j, \cdots, v_N\}$, where v_j represents the index of a Computation Node. The communication links between the networked computation nodes are represented by the set $E = \{e_{ij} \in \{0, 1\} | i = 1, 2, \cdots, M, j = 1, 2, \cdots, N\}$, where $e_{ij} = 1$ indicates that data transmission is possible between u_i and v_j; conversely, $e_{ij} = 0$ implies that no communication can occur.

2.1 Networked Computation Nodes Model

Networked computation nodes can be categorized as Task Nodes and Computation Nodes. The models for these two types of nodes are described below. 1) Task Node. A Task Node u_i is represented by the 5-tuple $(f_i, p_i, l_i, \mu_i, \alpha_i)$. In this tuple, f_i denotes the computational capability of the Task Node, and p_i denotes its transmit power. $l_i = \{x_i, y_i\}$ represents the coordinates of the Task Node, where x_i and y_i are its horizontal and vertical coordinates in a 2D plane, respectively. μ_i is a proportionality coefficient, representing the baseline level of power consumption for a given computational capability. α_i is the computation exponent, which describes the rate at which power consumption increases with computational capability. The energy consumption of the CPU processor can be estimated using the speed-power curve[19]. When the computation speed of the node is f_i, its speed-power curve is expressed as $P_i = \mu_i \cdot f_i^{\alpha_i}$. Each Task Node generates several compute-intensive real-time tasks. It is assumed that u_i generates K_i tasks in a time interval. The k-th task, t_{ik}, is represented by a 3-tuple $\{D_{ik}, C_{ik}, \text{Time}_{ik}\}$, where D_{ik} is the data volume of task t_{ik}, C_{ik} is the required computational workload, and Time_{ik} is its completion deadline. 2) Computation Node. A Computation Node v_j is represented by the 5-tuple $(f_j, \omega_j, l_j, \mu_j, \alpha_j)$. Here, f_j denotes the computational capability of the Computation Node, and ω_j denotes the number of its computation cores. $l_j = \{x_j, y_j\}$ represents its coordinates, where x_j and y_j are its horizontal and vertical coordinates, respectively. μ_j is the scaling factor, representing the baseline power consumption level of computing node v_j under a given computing power; α_j is the computing exponent, describing the rate at which the power consumption of computing node v_j increases with computing power. Together, they determine the power consumption model of the computing node. The speed-power curve of Computation Node v_j can be expressed as $P_j = \mu_j \cdot sv_j^{\alpha_j}$.

2.2 Latency Model

The real-time tasks generated by a Task Node can be executed in two ways: local computation or scheduling to another Computation Node. The scheduling decision for task t_{ik} can be represented by $X_{ikj} \in \{0, 1\}$. A value of $X_{ikj} = 1$ indicates that Task Node u_i schedules its k-th task to Computation Node v_j for execution, while $j = 0$ denotes local computation. Each task can only be processed by one Computation Node or locally, which means $\sum_{j=0}^{N} X_{ikj} = 1$.

If scheduling task t_{ik} from Task Node u_i to another Computation Node would not meet the deadline, the Task Node, driven by energy-saving considerations, may choose to execute the task locally. Local execution time of a task $\text{time}_{ik}^{\text{lexe}}$ specifically refers to the pure computation time of the k-th task of task node u_i when it is executed locally. In this case, $X_{ik0} = 1$, and its execution time is obtained from equation (1):

$$\text{time}_{ik}^{\text{lexe}} = C_{ik} / su_i \tag{1}$$

When Task Node u_i schedules task t_{ik} to Computation Node v_j for execution, such that $X_{ikj} = 1$, Orthogonal Frequency Division Multiple Access (OFDMA) is ad-

opted as the data transmission method[20]. The transmission rate R_{ikj} for a Task Node transmitting to a Computation Node can be obtained from equation (2).

$$R_{ikj} = B_j/\omega_j \cdot \log_2\left(1 + p_{ik} \cdot \rho \cdot \left\| l_i - l_j \right\|^{-Q} / \sigma^2\right) \tag{2}$$

In this equation, B_j is the network communication bandwidth of v_j, σ^2 is the Gaussian white noise power, ρ is the channel power gain per unit distance, σ is the path loss exponent, and $\|\cdot\|$ denotes the Euclidean norm. The transmit power for u_i to send t_{ik} to v_j is defined as $p_{ik} = D_{ik} \Big/ \sum_{k=1}^{K_i} D_{ik}$, the denominator $\sum_{k=1}^{K_i} D_{ik}$ represents the total data volume of all tasks. The total latency $\text{time}_{ikj}^{\text{nexe}}$ for scheduling task t_{ik} to v_j is given by equation (3):

$$\text{time}_{ikj}^{\text{nexe}} = \text{time}_{ikj}^{\text{tran}} + \text{time}_{ikj}^{\text{ncom}} \tag{3}$$

In this equation, $\text{time}_{ikj}^{\text{tran}} = D_{ik}/R_{ikj}$ is the transmission time from u_i to v_j, and $\text{time}_{ikj}^{\text{ncom}} = C_{ik}/f_j$ is the computation processing time after the task is transferred. The total execution latency $\text{time}_{ik}^{\text{exe}}$ for the k-th task t_{ik} of Task Node u_i is given by equation (4):

$$\text{time}_{ik}^{\text{exe}} = X_{ik0} \cdot \text{time}_{ik}^{\text{lexe}} + X_{ikj} \cdot \text{time}_{ikj}^{\text{nexe}} \tag{4}$$

System satisfaction, S, is defined as the percentage of the total number of tasks successfully executed before their deadlines to the total number of all tasks, which is expressed in equation (5):

$$S = \sum_{i=1}^{M} F_i \Big/ \sum_{i=1}^{M} K_i \tag{5}$$

In this equation, F_i represents the number of tasks from Task Node u_i that are successfully completed before their deadlines.

2.3 Energy Consumption Model

Based on the execution method, the energy consumption can be divided into two types: local computation energy and energy consumed for offloaded computation. When a task is computed locally, only local computation energy is incurred. Therefore, the energy consumed by Task Node u_i for computing task t_{ik} locally is shown in equation (6):

$$E_{ik}^{\text{lexe}} = P_i \cdot \text{time}_{ik}^{\text{lexe}} \tag{6}$$

The total energy consumption for Task Node u_i to schedule task t_{ik} for execution on Computation Node v_j is given by equation (7):

$$E_{ikj}^{\text{nexe}} = E_{ikj}^{\text{tran}} + E_{ikj}^{\text{ncom}} \tag{7}$$

In this equation, the transmission energy is $E_{ikj}^{\text{tran}} = p_{ik} \cdot \text{time}_{ikj}^{\text{tran}}$, and the computation energy is $E_{ikj}^{\text{ncom}} = pv_j \cdot \text{time}_{ikj}^{\text{ncom}}$. The total execution energy for the k-th task t_{ik} of Task Node u_i can be obtained from equation (8):

$$E_{ik}^{\text{exe}} = X_{ik0} \cdot E_{ik}^{\text{lexe}} + X_{ikj} \cdot E_{ikj}^{\text{nexe}} \tag{8}$$

2.4 Optimization Problem

This paper comprehensively considers a series of factors, including the heterogeneous communication and computation capabilities of networked computation nodes, the diversified computation requirements and real-time demands of tasks, and the energy consumption of each node. Subject to latency and resource constraints, the objective is to minimize task execution latency and system energy consumption while maximizing overall system satisfaction, thereby achieving an optimal balance between task latency and the energy consumption of Computation Nodes. The optimization problem can be formulated as shown in equation (9):

$$\begin{aligned}
&\min \lambda_1 \cdot [\max(\text{time}_{ik}^{\text{exe}})] + \lambda_2 \cdot [\sum_{i=1}^{M}\sum_{k=1}^{K_i} (\lambda_2 \cdot E_{ik}^{\text{exe}})] + \lambda_3 S^{-1} \\
&\text{s.t. } C_1: X_{ikj} \in \{0, 1\} \\
&C_2: \sum_{j=0}^{N} X_{ikj} = 1 \\
&C_3: \sum_{i=1}^{M}\sum_{k=1}^{K_i} X_{ikj} \leqslant \omega_j \\
&C_4: i \in \{1, 2, \cdots, M\}, j \in \{1, 2, \cdots, N\}, k \in \{1, 2, \cdots, K_i\}
\end{aligned} \tag{9}$$

In this formulation, λ_1, λ_2 and λ_3 are the weighting factors for latency, energy consumption, and system satisfaction, respectively, satisfying the condition $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Constraints C_1 and C_2 indicate that each task can only be executed either locally or on one Computation Node. Constraint C_3 ensures that the number of tasks served by a Computation Node does not exceed its number of idle computation cores.

3 A Real-Time Task Scheduling Algorithm Based on a Two-Sided Matching Game

Grounded in matching theory[21], this paper proposes a real-time task scheduling algorithm based on a two-sided matching game. This algorithm utilizes a matching game between Task and Computation Nodes to reach a Nash equilibrium. When a stable matching between the two parties is achieved, the total system cost is minimized. A real-time task matching model for Dispersed Computing is considered, which consists of two disjoint finite sets: a set of tasks, denoted as Task, and a set of Computation Nodes, denoted as NCP. Each element t_{ik} in the Task set has a specific preference for each

element v_j in the NCP set, denoted by $\text{pref}_{t_{ik}}(v_j)$. Similarly, each element v_j in the NCP set has a specific preference for each element t_{ik} in the Task set, denoted by $\text{pref}_{v_j}(t_{ik})$. When task t_{ik} is matched with Computation Node v_j, the decision variable X_{ikj} is set to 1.

3.1 The Preference List

In a matching algorithm, the preference list is a critical component that not only serves as the foundation for the algorithm's execution but also is key to achieving a fair and efficient matching outcome. The rule for calculating the preference value of a task t_{ik} for a Computation Node v_j is defined as shown in equation (10):

$$\text{pref}_{t\,ik}(v_j)=\begin{cases}0, & \text{if } e_{ij}=0 \text{ or } \text{time}_{ik}^{\text{exe}}>\text{Time}_{ik}\\ \gamma_1\cdot R_{ikj}+\gamma_2\cdot sv_j, & \text{else}\end{cases} \tag{10}$$

In this equation, γ_1 and γ_2 represent the preference weights that a task assigns to the distance and computational capability of a Computation Node, respectively, with the condition $\gamma_1+\gamma_2=1$. A task t_{ik} prioritizes Computation Nodes that offer higher transmission speeds and stronger computational capabilities, with the objective of minimizing execution time and reducing the energy consumption of its originating Task Node. The preference value is set to 0 if no communication link exists between the nodes or if the offloading execution time would exceed the task's completion deadline. By calculating the preference value $\text{pref}_{t_{ik}}(v_j)$ for t_{ik} to v_j according to equation (10) and sorting these values in descending order, the preference list for all tasks, denoted a spref, can be obtained.

To address the conflict in preference values between two computing nodes v_j and v_1 and their transmission rates $R_{ikj}=R_{ikl}$ and computing capabilities $sv_j=sv_l$, a secondary judgment rule is added: First, compare the energy consumption efficiency per unit computing power of the two nodes $\text{eff}_v=\dfrac{p_v}{sv}$, $p_v=\mu_v\cdot v^{\alpha}$ is node power consumption. Select the smaller eff_v node to reduce system energy consumption; if the energy consumption efficiency is equal, then further compare the number of idle cores w_v of the nodes, and prioritize the larger w_v node to avoid delays in subsequent tasks due to resource overload, ensuring that the task's preference selection for computing nodes is more comprehensive and in line with the optimization goals.

Defined eco_{ikj} as the energy consumption requirement per unit load on a computing node v_j for a task t_{ik}, it quantifies the degree to which task execution consumes the node's energy. The calculation formula is:

$$\text{eco}_{ikj}=E_{ikj}^{\text{nexe}}/C_{ik} \tag{11}$$

The smaller the value of the computational load C_{ik} for a task is, the less energy the task can consume to complete the unit computational load, and the less energy pressure it puts on the node. The rule for calculating the preference value of a Computation Node v_j for a task t_{ik} is expressed in equation (12):

$$\text{pref}v_j(t_{ik})=\eta_1\cdot C_{ik}^{-1}+\eta_2\cdot\text{Time}_{ik}^{-1}+\eta_3\cdot\text{eco}_{ikj}^{-1} \tag{12}$$

In this formula, η_1, η_2 and η_3 are the preference weights that a Computation Node assigns to the task's computational resource demand, its completion deadline and low energy consumption requirements, respectively, under the condition $\eta_1+\eta_2+\eta_3=1$. A Computation Node shows a higher preference for tasks that require fewer computational resources and have more urgent real-time requirements. This strategy allows the node to satisfy the task's execution demands while lowering its own energy consumption. By calculating the preference value $\text{pref}_{v_j}(t_{ik})$ based on equation (12) and sorting these values in descending order, the preference list for the Computation Node, ncp_j, is generated.

3.2 Two-Sided Matching-Based Task Scheduling Algorithm

After a task has generated its preference list for Computation Nodes, if the list is empty, the task is processed locally. Otherwise, the task submits an execution request to the top-ranked Computation Node in its preference list. The Computation Node, in turn, generates its own preference list for all tasks from which it has received requests. A decision to either accept or reject a task is then made based on this preference list and its own resource availability, thereby achieving a many-to-many stable matching between tasks and Computation Node. The proposed real-time task scheduling algorithm, which is based on a two-sided matching game, takes the sets Task, NCP, and E as input, and produces the scheduling decision X as output. The specific steps are detailed as follows:

1) Initialize system parameters. The unmatched task list and the preference list for each Computation Node are initialized as empty. The scheduling decision for all tasks is set to 0. Subsequently, the preference list for every task is generated according to equation (10).

2) Add all tasks from all Task Nodes to the unmatched task list.

3) For each task in the unmatched task list, a prefer-

ence list of Computation Node is generated according to equation (11). If the preference list for a task is empty, it is scheduled for local execution. Otherwise, the task proposes an execution request to its most preferred Computation Node. Upon receiving the request, Computation Node adds the proposing task to its own preference list. Once all tasks have made their proposals, each Computation Node generates and sorts its preference list in descending order. This signifies that every task has made a locally optimal scheduling decision based on its self-interest. So the unmatched task list is cleared.

4) For each Computation Node v_j, if the number of received proposals is less than its number of idle cores (ω_j), it accepts all proposals. Otherwise, it accepts the top ω_j tasks from its sorted preference list. For each accepted task t_{ik}, set its scheduling decision $X_{ikj}=1$. The remaining proposing tasks are rejected.

5) Add all rejected tasks back to the unmatched task list. Repeat Steps 3 and 4 until the unmatched task list is empty. At this point, all tasks have been stably matched, and the algorithm terminates.

4 Experiments and Analysis

4.1 Experiments Setting

The performance of the real-time task scheduling algorithm based on a two-sided matching game was evaluated through simulation experiments. A 1 000 m × 1 000 m two-dimensional planar area was considered, within which multiple networked computation nodes were uniformly distributed. These nodes cooperated to complete a series of compute-intensive real-time tasks. Furthermore, to verify the advantages of Dispersed Computing in comparison with local computation and edge computing, a fixed-position edge server was introduced to simulate the edge computing paradigm within the area. The specific parameters used for the simulation experiments in this section were adopted from Refs. [22-25]. The computing power $su_i=1.0$ - 2.5 GHz of the task node u_i, the computing power $sv_j=2.0$ - 4.0 GHz of the compute node v_j, and the number of idle cores $w_j=2$ - 4. The task characteristics are as follows: $D_{ik}=1$ - 10 MB, $C_{ik}=1\times10^8$ - 5×10^8 CPU cycles, $\text{Time}_{ik}=0.5$ - 2.0 s. The algorithm's preference weights were optimized to be $\gamma_1=0.55$, $\gamma_2=0.45$ and $\eta_1=0.3$, $\eta_2=0.4$, $\eta_3=0.3$.

4.2 Comparison of Results

The real-time task scheduling algorithm based on a two-sided matching game proposed in this paper is denoted as RT-TSMG. To evaluate the effectiveness of RT-TSMG, three common task scheduling algorithms were introduced as benchmarks for comparative analysis. These are:

Random Scheduling (RAN): In this algorithm, a task randomly selects an available Computation Node for execution.

Heterogeneous Earliest Finish Time (HEFT) [26]: This is a policy that, based on task priorities, schedules tasks in a heterogeneous computing environment to minimize the total completion time. In the simulation environment designed here, this algorithm prioritizes tasks that are expected to finish the earliest.

Genetic Algorithm (GA) [27]: This algorithm iteratively evolves a population of candidate solutions through crossover, mutation, and selection operations to produce approximate solutions to the problem. In our experiments, the GA used equation (9) as its fitness function for population iteration.

1) Total task execution cost

The comparison of the total task execution cost between RT-TSMG and the benchmark algorithms under a varying number of users is shown in Fig. 1. It was observed that the total system cost increased with the number of task nodes. This is because a larger number of users lead to more tasks requiring execution, which in turn increases both task execution time and total system energy consumption. By employing an efficient task scheduling rule, RT-TSMG achieved average savings of 8.4%, 3.7%, and 23.2% in total execution cost compared to HEFT, GA, and RAN, respectively. This demonstrates that RT-TSMG significantly improves the system's scalability and efficiency.

2) Task execution time and energy consumption

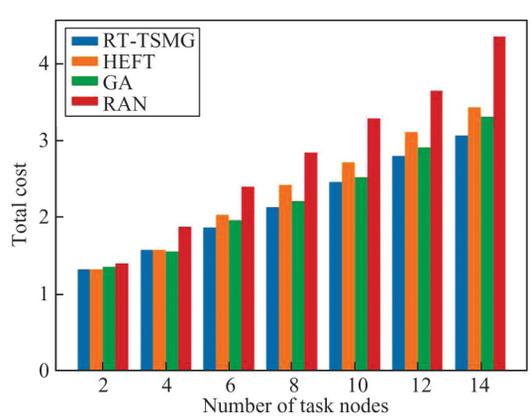

Fig.1 Comparison graph of total cost of task execution

The detailed trends of task execution latency and energy consumption with an increasing number of task nodes are depicted in Fig. 2. As shown in Fig. 2(a), the task execution latency for RT-TSMG, HEFT, and GA exhibited an upward trend as the number of task nodes increased. The reason is that the growing number of tasks intensified resource competition, causing some tasks to be allocated to nodes with lower computational capabilities. In contrast, the execution latency of the RAN scheme, which selects nodes randomly, tended towards a stable and balanced state after an initial rise. Furthermore, RT-TSMG fully considers the heterogeneity of task resource demands during its scheduling process. This allows it to rationally prioritize tasks with higher resource requirements to resource-abundant nodes, which significantly reduces the maximum execution latency when all tasks are considered. RT-TSMG achieved average latency savings of 9.5%, 14.3%, and 26.9% compared to HEFT, GA, and RAN, respectively. As shown in Fig. 2(b), the task execution energy consumption also increased with the number of task nodes, and this growth trend became more pronounced in later stages due to intensified resource competition. In this process, the energy consumption of RT-TSMG was slightly higher than that of GA. This is because GA does not consider the balance between task latency and energy consumption, whereas RT-TSMG fully considers balancing the interests of tasks and computation nodes. Consequently, while achieving excellent latency performance, its energy consumption was slightly higher than that of GA. However, it still achieved energy savings of 9.1% and 20.5% compared to HEFT and RAN, respectively.

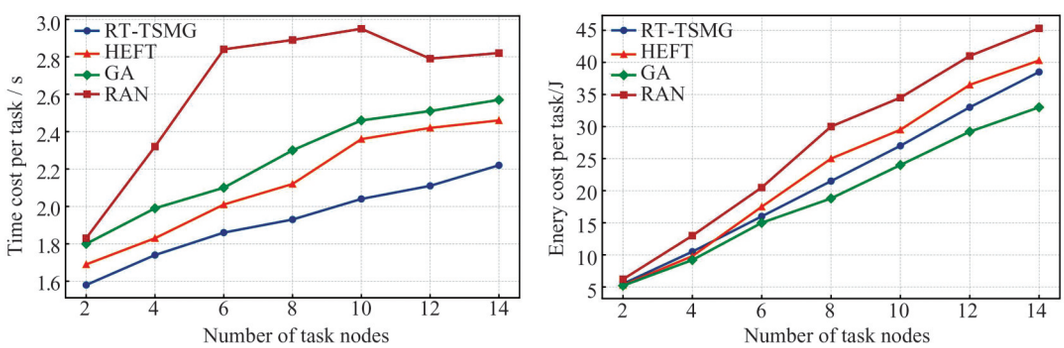

Fig.2 Comparison graphs of task execution time and energy consumption

3) System satisfaction

Figure 3(a) shows the trend of system satisfaction with the number of task nodes under different scheduling algorithms, reflecting the proportion of tasks completed before the deadline. As the number of task nodes increases, the system satisfaction of all four algorithms decreases. This is because, with the expansion of task scale, the real-time requirements of some tasks are difficult to meet. Among them, RT-TSMG consistently maintains the best system satisfaction, remaining above 79% when the number of task nodes is no more than 8. Even when the number of task nodes increases to 14, its satisfaction is still significantly higher than HEFT, GA, and RAN. Compared with HEFT, GA, and RAN, RT-TSMG's system satisfaction is improved by an average of 6.7%, 15.2%, and 27.1%, respectively. This result verifies the effectiveness of RT-TSMG in achieving precise alignment between tasks and computing nodes through bilateral matching game theory, demonstrating a significant advantage in ensuring task real-time performance.

4) System throughput

The comparison of system throughput is shown in Fig. 3(b). Throughput is defined as the average number of tasks completed by the system per unit of time. It can be observed from Fig. 3(b) that RT-TSMG significantly outperformed the other schemes in terms of system throughput. This is primarily attributed to RT-TSMG's ability to meticulously consider the time requirements of each task and perform corresponding scheduling optimizations based on task urgency, with the aim of shortening the makespan of all tasks and thereby boosting system throughput. Compared to RAN, HEFT, and GA, RT-TSMG achieved throughput improvements of 67.1%, 23.1%, and 28.8%, respectively. The HEFT scheme may sometimes postpone the scheduling of tasks with longer execution times, while GA, despite optimizing overall system utility, gave slightly less consideration to task latency, resulting in a performance inferior to that of RT-TSMG.

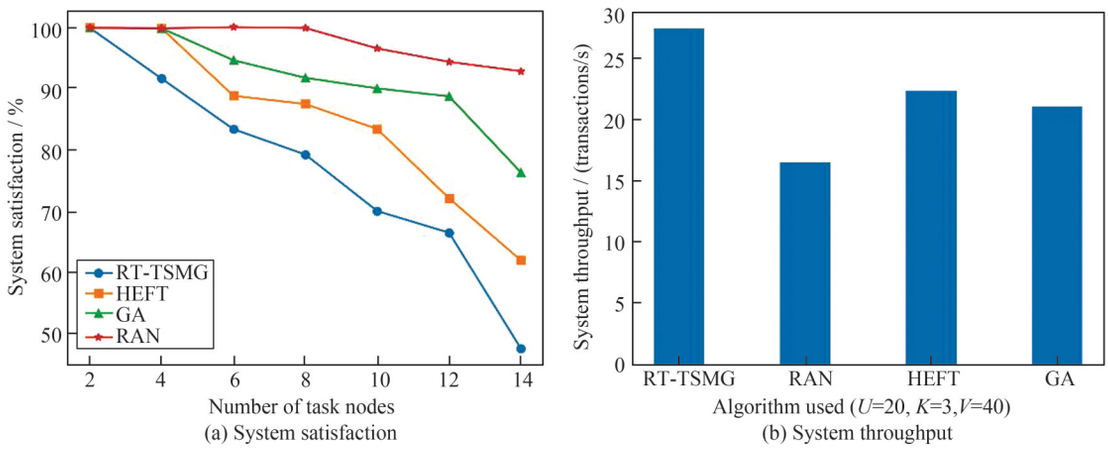

Fig.3 Comparison graphs of system satisfaction and system throughput

5 Conclusion

To address the unbalanced trade-off between task execution latency and energy consumption in dispersed computing environments, this paper constructed a real-time task scheduling model for such environments. This model is designed to minimize task execution latency and system energy consumption while maximizing overall system satisfaction, under the constraint of ensuring real-time task execution. To obtain a task scheduling strategy, a real-time task scheduling algorithm based on a two-sided matching game was designed. The simulation results demonstrate that the proposed algorithm achieved average savings of 8.4%, 3.7%, and 23.2% in total execution cost compared to the HEFT, GA, and RAN schemes, respectively, and effectively balanced the execution latency of tasks with the energy consumption of Computation Nodes.

Future work will focus on designing a rational incentive mechanism to incentivize privately-owned devices to voluntarily contribute their idle resources. Furthermore, it is imperative to construct secure node interaction mechanisms and reliable communication protocols.

References

[1] Schurgot M R, Wang M, Conway A E, *et al*. A dispersed computing architecture for resource-centric computation and communication[J]. *IEEE Communications Magazine*, 2019, **57**(7): 13-19.

[2] Wu H J, Liu F, Liu B, *et al*. Dispersed computing: Technologies, applications and challenges[J]. *Journal of Frontiers of Computer Science and Technology*, 2020, **14**(5): 721-730(Ch).

[3] Sun Y X. *Enhancing Anonymity Systems Under Network and User Dynamics*[D]. Princeton: Princeton University, 2020.

[4] Michel O, Sonchack J, Keller E, *et al*. Packet-level analytics in software without compromises[C]// *USENIX Workshop on Hot Topics in Cloud Computing* (*HotCloud* 2013). Santa Clara: USENIX Association, 2013: 1-6.

[5] Zaki Y, Pötsch T, Ahmad T. Application-level congestion control: overcoming TCP's limitations in cellular networks [C]// *Proceedings of the* 2018 *USENIX Annual Technical Conference* (*USENIX ATC* 2018). Boston: USENIX Association, 2018: 1-14.

[6] Hui H W. *Research on Cybersecurity Model and Algorithm in Dispersed Computing Environment*[D]. Beijing: University of Science and Technology Beijing, 2023(Ch).

[7] Ghosh P, Nguyen Q, Krishnamachari B. Container orchestration for dispersed computing[C]//*Proceedings of the 5th International Workshop on Container Technologies and Container Clouds.* New York: ACM, 2019: 19-24.

[8] Yang H G, Li G, Sun G Y, *et al*. Dispersed computing for tactical edge in future wars: Vision, architecture, and challenges[J]. *Wireless Communications and Mobile Computing*, 2021, **2021**(1): 8899186.

[9] An X M, Fan R F, Hu H, *et al*. Joint task offloading and resource allocation for IoT edge computing with sequential task dependency[J]. *IEEE Internet of Things Journal*, 2022, **9**(17): 16546-16561.

[10] Li Y H, Jiang C S. Distributed task offloading strategy to low load base stations in mobile edge computing environment[J]. *Computer Communications*, 2020, **164**: 240-248.

[11] Song S D, Ma S Y, Zhao J M, *et al*. Cost-efficient multi-service task offloading scheduling for mobile edge computing[J]. *Applied Intelligence*, 2022, **52**(4): 4028-4040.

[12] Kuang Z F, Chen Q L, Li L F, *et al*. Multi-user edge computing task offloading scheduling and resource allocation based

on deep reinforcement learning[J]. *Chinese Journal of Computers*, 2022, **45**(4): 812-824(Ch).

[13] Sacco A, Esposito F, Marchetto G, *et al*. Sustainable task offloading in UAV networks *via* multi-agent reinforcement learning[J]. *IEEE Transactions on Vehicular Technology*, 2021, **70**(5): 5003-5015.

[14] Seid A M, Boateng G O, Mareri B, *et al*. Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network[J]. *IEEE Transactions on Network and Service Management*, 2021, **18**(4): 4531-4547.

[15] Niu Z C, Liu H, Lin X M, *et al*. Task scheduling with UAV-assisted dispersed computing for disaster scenario[J]. *IEEE Systems Journal*, 2022, **16**(4): 6429-6440.

[16] Poylisher A, Cichocki A, Guo K, *et al*. Tactical Jupiter: Dynamic scheduling of dispersed computations in tactical MANETs[C]//*MILCOM* 2021 *IEEE Military Communications Conference* (*MILCOM*). New York: IEEE, 2021: 102-107.

[17] Yang C S, Avestimehr A S, Pedarsani R. Communication-aware scheduling of serial tasks for dispersed computing[C]// 2018 *IEEE International Symposium on Information Theory* (*ISIT*). New York: IEEE, 2018: 1226-1230.

[18] Hu D Y, Krishnamachari B. Throughput optimized scheduler for dispersed computing systems[C]//2019 7*th IEEE International Conference on Mobile Cloud Computing*, *Services*, *and Engineering* (*MobileCloud*). New York: IEEE, 2019: 76-84.

[19] Wang Q, Mei X X, Liu H, *et al*. Energy-aware non-preemptive task scheduling with deadline constraint in DVFS-enabled heterogeneous clusters[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2022, **33**(12): 4083-4099.

[20] Wang Y Y, Huang J R. Efficient space-time signal processing scheme of frequency synchronization and positioning for sensor networks[J]. *Sensors*, 2023, **23**(4): 2115.

[21] Yang G S, Wang B Y, He X Y, *et al*. Competition-congestion-aware stable worker-task matching in mobile crowd sensing [J]. *IEEE Transactions on Network and Service Management*, 2021, **18**(3): 3719-3732.

[22] Ma G F, Li H R, Wang X W, *et al*. Mobility-aware task splitting and computation resource allocation for distributed multi-access edge computing enabled vehicular network[C]//2021 *International Conference on Mechanical*, *Aerospace and Automotive Engineering.* New York: ACM, 2021: 164-170.

[23] Wang G Y, Yu X B, Xu F C, *et al*. Task offloading and resource allocation for UAV-assisted mobile edge computing with imperfect channel estimation over Rician fading channels[J]. *EURASIP Journal on Wireless Communications and Networking*, 2020, **2020**(1): 169.

[24] Guo K, Gao R F, Xia W C, *et al*. Online learning based computation offloading in MEC systems with communication and computation dynamics[J]. *IEEE Transactions on Communications*, 2021, **69**(2): 1147-1162.

[25] Chen C, Zeng Y N, Li H, *et al*. A multihop task offloading decision model in MEC-enabled Internet of vehicles[J]. *IEEE Internet of Things Journal*, 2023, **10**(4): 3215-3230.

[26] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. *IEEE Transactions on Parallel and Distributed Systems*, 2002, **13**(3): 260-274.

[27] Holland J H. Genetic algorithms[J]. *Scientific American*, 1992, **267**(1): 66-73.

分散计算环境下基于双边匹配博弈的实时任务调度算法

李硕[1]，方祖应[2]，周国强[2]，戴桂兰[3†]

1. 黄淮学院 智能制造学院，河南 驻马店 463000
2. 南京邮电大学 计算机科学学院，江苏 南京 210023
3. 清华大学 计算机科学与技术系，北京 100084

摘要：在万物互联时代，分散计算通过整合异构设备空闲资源缓解了终端计算能力不足的问题，但任务执行延迟与节点能耗的博弈不平衡、设备异构性带来的调度适配难题亟待解决。针对该问题，本文构建了兼顾任务实时性、执行延迟、系统能耗与节点利益的多目标实时任务调度模型，以最小化延迟上限与总能耗、最大化系统满意度为优化目标，提出了基于双边匹配博弈的实时任务调度算法。通过设计任务-计算节点双向偏好机制，结合多轮次稳定匹配策略，实现任务与节点的精准适配；同时引入次级判断规则解决偏好冲突，保障调度公平性与效率。仿真实验表明，与基线相比，该算法显著降低了总执行成本，有效平衡了任务执行延迟和计算节点的能耗，并兼顾了网络中每个计算节点的利益。

关键词：分散计算；实时任务；任务调度；双边匹配博弈

□