Computing
https://doi.org/10.1007/s00607-023-01160-2

REGULAR PAPER

Many-to-many matching based task allocation for dispersed computing

Hongwen Hui[1] · Fuhong Lin[1,2] · Lei Meng[1] · Lei Yang[3] · Xianwei Zhou[1]

Received: 18 March 2022 / Accepted: 10 January 2023

Abstract
Dispersed computing is a new resource-centric computing paradigm, which makes use of idle resources in the network to complete the tasks. Effectively allocating tasks between task nodes and networked computation points (NCPs) is a critical factor for maximizing the performance of dispersed computing. Due to the heterogeneity of nodes and the priority requirements of tasks, it brings great challenges to the task allocation in dispersed computing. In this paper, we propose a task allocation model based on incomplete preference list. The requirements and permissions of task nodes and NCPs are quantitatively measured through the preference list. In the model, the task completion rate, response time, and communication distance are taken as three optimizing parameters. To solve this NP-hard optimization problem, we develop a new many-to-many matching algorithm based on incomplete preference list. The unilateral optimal and stable solution of the model are obtained. Taking into account the needs for location privacy-preserving, we use the planar Laplace mechanism to produce obfuscated locations instead of real locations. The mechanism satisfies ε-differential

✉ Fuhong Lin
FHLin@ustb.edu.cn

Hongwen Hui
hhw21788712@163.com

Lei Meng
anthemmong@qq.com

Lei Yang
leiy@unr.edu

Xianwei Zhou
xwzhouli@sina.com

1 Department of Communication Engineering, University of Science and Technology Beijing (USTB), Beijing 100083, China

2 Shunde Graduate School of University of Scienc and Technology Beijing, Guangdong 528300, China

3 Department of Computer Science and Engineering, University of Nevada at Reno, Reno, State 89557, USA

Published online: 25 January 2023

privacy. Finally, the efficacy of the proposed model is demonstrated through extensive numerical analysis. Particularly, when the number of task nodes and NCPs reaches 1:2, the task completion rate can reach 99.33%.

Keywords Dispersed computing · Networked computation points · Task allocation · Location privacy-preserving · Many-to-many matching

Mathematics Subject Classification 49

1 Introduction

1.1 Motivation

Dispersed computing describes a new computing paradigm that can perform computation in the most suitable location according to task requirements, which can greatly reduce communication and computing costs [1]. A dispersed computing systems consists of a group of highly heterogeneous networked computation points (NCPs) (such as smart phones, tablets, internet of things terminal servers, base stations, and users computers) that can provide computing services [2]. Compared with other computing paradigms (cloud computing, fog computing, and edge computing), the advantage of dispersed computing lies in the flexibility of choosing the most convenient location for the mobile devices to carry out the computation [3–5], which relies on efficient task allocation in dispersed computing. There are many challenges to the task allocation of dispersed computing since the highly dynamic and unstable of dispersed computing environment(such as the heterogeneity in computation power and communication bandwidth). Among them, one of the most important challenge for the task allocation is how to quickly identify and deal with the instantaneous changes of the network.

In dispersed computing systems, the computing capabilities and types of computation that can be performed by different nodes are vary greatly. Strong nodes (such as powerful servers and base stations) may be able to efficiently complete various complex tasks, whereas weak nodes (such as smart phones and tablets) may perform some simple tasks. The communication bandwidth between different nodes is limited and heterogeneous. In addition, in many application scenarios of dispersed computing, tasks issued by task nodes are usually set with precedence requirements (captured by a Directed Acyclic Graph), besides the requirements, such as computing power, delay, and throughput [6–8]. Thus, the successful application of dispersed computing depends closely on the design of efficient task allocation mechanisms, which requires carefully consider the heterogeneity of computation and communication. To design an effective task allocation mechanism suitable for dispersed computing environment, the following three major challenges must be addressed:

- *Heterogeneity* The nodes and communication bandwidth in dispersed computing are highly heterogeneous, and the tasks of different nodes are very different. To establish a task allocation mechanism in a heterogeneous network, it must be considered that some NCPs are unable to complete all types of tasks. For example,

high-throughput matrix operations have high requirements on the graphics processing unit (GPU) of the computing device, and nodes with weak computing power cannot complete such tasks. In the dispersed computing, some task nodes require multiple NCPs to cooperate to complete their tasks, while some NCPs can handle the tasks of multiple task nodes at the same time. In other words, task allocation in this environment is a many-to-many relationship, not a one-to-one relationship.

- *Preference* There is no fixed relationship between task nodes and the NCPs in the dispersed computing environment. The task nodes and the NCPs that complete the task establish a provisional cooperative relationship. Therefore, task nodes need to explicitly attach some necessary conditions to ensure the quality of task completion. Similarly, NCPs also have the right to choose their own preferred tasks or reject some tasks. It is an important challenge that the designed task allocation mechanism needs to have stronger compatibility and scalability. This mechanism allows nodes to choose partners based on more objective needs and preferences, thereby encouraging more NCPs to participate in computation.
- *Location privacy-preserving* The dispersed computing is a highly open computing paradigm that connects all devices in the word that can perform computing. Location-based service and efficient task allocation usually requires the location information of nodes. The increasing exposure of user location information raises important privacy issues. Therefore, the location privacy protection of nodes is a important factor that must be considered in task allocation for dispersed computing. A novel task allocation mechanism is designed, which not only takes full advantage of the convenience brought by location information, but also protects the privacy information of nodes.

1.2 Related work

1.2.1 Dispersed computing

The conceptual roots and ideas of dispersed computing first came from García-Valls's work [9], and the authors identified a new computing paradigm that was called "social dispersed computing". Dispersed computing has a unique advantage that it can mission-aware and effectively use geographically dispersed and idle computing resources, which called "dispersed mission-aware computation" [1]. Some preliminary research results have been achieved, mainly on the discussion of architecture concepts and task scheduling. Yang et al. [5] discussed in detail the four different computing architectures (mobile cloud computing, fog computing, mobile edge computing, and mobile ad hoc network) and pointed out the limitations of these computing paradigms in the tactical environment, which implies the necessity of proposing dispersed computing. As a new computing paradigm, related research is still in its infancy, and it is necessary and urgent to carry out research on broader topics (i.e., data security, privacy protection, task allocation, intrusion detection, etc.) in dispersed computing environment.

1.2.2 Task allocation in dispersed computing

Some preliminary explorations are made task allocation in dispersed computing environment. Yang et al. [2] studied the task scheduling problem of heterogeneous network, proposed a max-weighted scheduling strategy for dispersed computing, and proved that the strategy is throughput-optimal by using Lyapunov method. Hu et al. [10] proposed a task scheduler based on two techniques (task duplication and task splitting) to improve the task throughput of dispersed computing systems. Zhou et al. [11] proposed a task-resource joint management model with state feedback control capability by using the advantages of resource awareness in dispersed computing, and proved that the stability of the system can be guaranteed in the mechanism. Ghosh et al. [12] proposed a container orchestration architecture, and used a directed acyclic graph (DAG) to distribute computation tasks to a set of network NCPs.

However, all these works ignore the heterogeneity of dispersed computing environment (i.e., differences of node's computing power and tasks). The assumption that each server can handle any type of task does not in line with the actual processing power of nodes in a heterogeneous environment. In addition, compared to throughput, task completion rate is insufficient studied in task allocation of dispersed computing. Nonetheless, task completion rate is a critical metric for studying task allocation of highly dynamic scenarios.

1.2.3 Task allocation in mobile crowdsensing scenarios

There are similarities for task allocation in dispersed computing scenarios and in mobile crowdsensing scenarios (MCS) [13–16]. They can all be attributed to a matching relationship between the two groups (task publishers or task nodes, and workers or NCPs). Two main modes of task allocation are proposed in MCS, namely, worker selected task (WST) and server assigned task (SAT) [17–19]. WST model is user-centric, and mobile users can choose tasks according to their location and preferences. SAT model is platform-centric, the platform collects the tasks released by the task requester and the information of all participants, which matches and distributes the tasks and participants in order to maximize the benefits of the platform, such as maximizing the task completion rate and minimizing the platform cost [20, 21].

By contrast, task allocation in dispersed computing differs from MCS in three aspects. First, the task allocation in MCS is mainly from the perspective of the platform, with the goal of maximizing platform revenue. It lacks a task selection strategy that considers the participants from the perspective of the participants. The dispersed computing is a non-central distributed system designed to incentive all idle resources to participate in computing. Second, the dispersed computing network is highly heterogeneous and dynamics. The computing power of nodes, the communication bandwidth between nodes, and the types of tasks are very different. Third, tasks in MCS are usually offline tasks, and most of the tasks involved in dispersed computing are complex online computing tasks (such tasks are more sensitive to network delay).

1.3 Main contributions

In this paper, a task allocation model based on incomplete preference list in dispersed computing is established, which is a multi-objective optimization model. The model comprehensively consider the three optimization objectives of task completion rate, response time, and communication distance. The incomplete preference list between task nodes and NCPs is introduced to fully reflect the heterogeneity of nodes and the difference of tasks, as well as the requirements and authority of both parties task nodes and NCPs in task allocation. The coupling of the preference list and the optimization problem induce great challenges for solving the task allocation model. We transform a task allocation problem into a many-to-many matching with incomplete preference list. In addition, the Planar Laplace mechanism is used to protect location privacy information of nodes. The mechanism satisfies ε-differential privacy protection and ε-geo-indistinguishability. The main contributions can be summarized as follows:

1. We establish a location privacy-preserving task allocation model is based on incomplete preference list to tackle the challenge due to the heterogeneity of nodes and the priority requirements of tasks. This model allows task nodes and NCPs to actively make choices according to their own preferences, mobilizing the enthusiasm of nodes to participate in joint computing.
2. We propose a many-to-many matching algorithm to solve the task allocation problem that is a Mixed Integer Non-Linear Programming (MINLP) problem. The algorithm is generalizable and can provide a new method for solving NP-hard.

The remainder of this paper is organized as follows. An overview of system model is presented in Sect. 2. The task allocation matching algorithm and relevant theoretical analysis are presented in Sect. 3. Experimental simulations are discussed in Sect. 4. Finally, the conclusion is made in Sect. 5.

2 System model and problem formulation

2.1 Problem description and mathematical formulation

We consider a task allocation process in dispersed computing scenario, which consists of task nodes (allocating tasks) and NCPs (processing tasks), as shown in Fig. 1. In this process, dispersed mission-aware computation and protocol stacks aims to jointly manage computing and network resources, and share mission details through new algorithms and protocols [1], which is the core in the dispersed computing architecture. When computing tasks appear, the task allocation algorithm uses the idle computing resources in the network to divide the tasks, and provides services to users in a way of collaboration and sharing of NCPs. Generally, the arrival of task nodes and NCPs obey a certain random process (such as a Poisson process [22–24]). The process of task allocation is carried out in batches. When the number of task nodes and NCPs reaches a certain threshold, the current batch of allocation will begin immediately. The nodes that arrive later are boiled down to the next batch of task allocation.

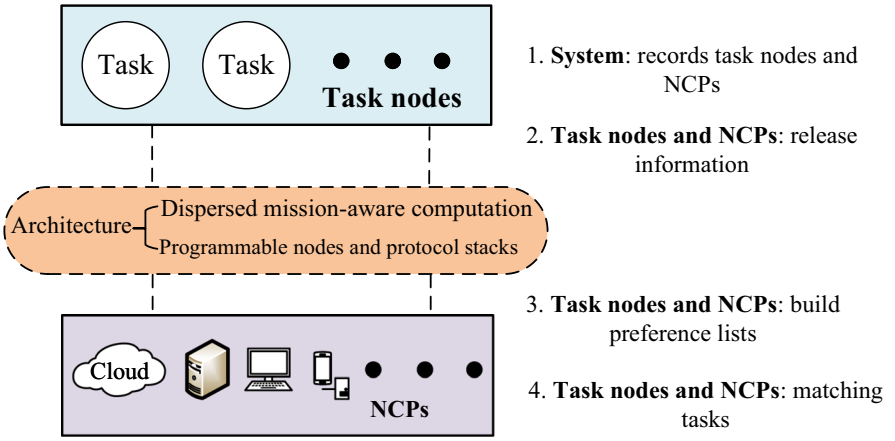

Fig. 1 Illustration of task allocation

The dispersed computing paradigm is suitable for applications in different scenarios, including scientific research fields, military combat scenarios, daily life needs, etc. The tasks involved range from complex mathematical or engineering problems (completing such tasks requires significant computing resources) to simple single-threaded tasks. That is, tasks in a dispersed computing environment are very heterogeneous. Task allocation in different scenarios can be attributed to the matching problem between task nodes and NCPs. It may be that one task node needs multiple NCPs, or one NCP may complete the tasks of multiple task nodes.

Assuming that the system records the arrival of m task nodes (the node that publishes the task) in a batch and denoted by a set $\Omega \triangleq \{\omega_1, \omega_2, \cdots, \omega_m\}$, where ω_i represents the serial number of the task node. The type and size of tasks may be very different due to the heterogeneity of nodes. At the same time, the system perceives n NCPs (the node that processes the task) during computing power the time period denoted by a set $C \triangleq \{c_1, c_2, \cdots, c_n\}$, where c_j represents the serial number of the NCP. These NCPs have different due to the heterogeneity of nodes.

The main purpose of our work is to allocate all tasks of m task nodes to n NCPs reasonably and efficiently through the information exchange of middleware. Each task node ω_i reports its own actual location l_{ω_i}, a requirement of computing power c_{ω_i}, and the size of the task amount (denoted z_{ω_i}, where z_{ω_i} represents the total task of task node ω_i divided into z_{ω_i} indivisible small units (jobs)). Similarly, each NCP c_j reports its own actual location l_{c_j}, a feature of computing power c_{c_j}, and the size of the amount of calculation that can be provided (denoted z_{c_j}). In addition, each node only reports the location privacy protection version during the task allocation process to protect its location privacy information.

Figure 1 illustrates a decentralized architecture with direct communication between task nodes and NCPs. The tasks refer to online computing tasks, and each task has requirements for computing power, maximum allowable delay, and deadline. First, the dispersed computing system records task nodes and NCPs within a period of time

through a unique mission-aware computation algorithm [1]. The workflow of the task allocation is summarized as follow:

- The task nodes and NCPs release information based on their actual needs, including their additional conditions with preferences. Then, according to the released information, each node can calculate the priority order of its partners, which is called a preference list.
- Then, task nodes and NCPs start the task allocation process based on the incomplete preference list.

The task allocation model needs to satisfy the following characteristics:

1. Task node can add deadline and completion progress as requirements for each task.
2. The task node will reject the NCP whose distance exceeds its allowable range and computing power does not meet its requirements.
3. The NCP will reject the task node whose computing power does not meet its requirements.

These features guarantee the quality of task completion, even if there are a small number of tasks that are not completed in the first batch allocation. This is the characteristic of our task allocation scheme. In addition, the task allocation process is about time continuity, even if there are unfinished tasks in one batch allocation, it will automatically be returned to the next batch allocation. In this paper, the emphasis is laid on the task allocation within one batch, and each batch task allocation process is called a matching [25–27], denoted by μ. The percentage of the number of completed jobs in the total number of jobs is called the task completion rate for the matching μ, i.e.,

$$F(\mu) = \frac{\sum_{i=1}^{m} z'_{\omega_i}}{\sum_{i=1}^{m} z_{\omega_i}}, \tag{1}$$

where z'_{ω_i} represents the jobs successfully allocated by task node ω_i. Next, we will establish a task allocation model. The optimization objectives include: task completion rate, communication distance, and response time as follows, respectively

$$\begin{aligned} J_1 &= F(\mu) = \frac{\sum_{i=1}^{m} z'_{\omega_i}}{\sum_{i=1}^{m} z_{\omega_i}}, \\ J_2 &= \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} d(l_{\omega_i}, l_{c_j}), \\ J_3 &= Time_1 - Time_0. \end{aligned} \tag{2}$$

where $Time_1$ and $Time_e$ represent the start time and end time of the task allocation, respectively. Then the task allocation model based on preference list is represented as follows:

$$
\begin{aligned}
&\max J = J_1 + J_2^{-1} + J_3^{-1} \\
&s.t. \quad \sum_{i=1}^{m} x_{ij} \le z_{c_j}, \quad j = 1, 2, \cdots, n, \\
&\quad \sum_{j=1}^{n} x_{ij} \le z_{\omega_i}, \quad i = 1, 2, \cdots, m, \\
&\quad x_{ij} \in \{0, 1\}, \quad i = 1, 2, \cdots, m, \quad j = 1, 2, \cdots, n, \\
&\quad \text{if } x_{ij} = 1, \text{ then } \omega_i \in P(c_j) \text{ and } c_j \in P(\omega_i),
\end{aligned} \tag{3}
$$

where $x_{ij} = 1$ represents the formation of the task allocation between ω_i and c_j. The last constraint $P(c_j)$ and $P(\omega_i)$ respectively represent the preference list of c_j and ω_j, and the calculation method will be given in detail in Sect. 2.2. A preference list is more generic than a constraint which can combine additional qualitative measures extracted from the objective requirement to task nodes and NCPs. Therefore, the task allocation model (3) is called "a task allocation model based on preference list". In the task allocation model (3), the first and second constraints use symbols of less than or equal to, which means that maybe some tasks that are not allocated, and the resources of the NCPs are still idle. In fact, the efficiency of task allocation can be improved by controlling the ratio of the number of task nodes and NCPs.

2.2 Preference list

Next, we establish a preferences list is between task nodes and NCPs, and then built a task allocation model based on the list. Since the process involves some mathematical symbols in matching theory, for convenience sake, throughout this paper, three symbols are introduced in this paper, and their meanings are as follows:

1. $\omega_i \to c_j$, which means that node ω_i sends a request to node c_j;
2. $f_{\omega_i \to c_j}$, which represents the degree of preference of node ω_i to node c_j under rule f;
3. $\omega_i \leftrightarrow c_j$, which means that node ω_i and c_j form a cooperative relationship (or form a matching pair).

A calculation rule of preference list (about task node ω_i to NCP c_j) is defined as follows:

$$
f_{\omega_i \to c_j}((l_{\omega_i}, c_{\omega_i}, z_{\omega_i}), (l_{c_j}, c_{c_j}, z_{c_j})) = \begin{cases} 0, \text{ if } c_{\omega_i} \ge c_{c_j} \text{ or } d(l_{\omega_i}, l_{c_j}) \ge r_{\omega_i}, \\ \dfrac{\alpha}{d(l_{\omega_i}, l_{c_j})} + (1 - \alpha)(c_{c_j} - c_{\omega_i}), \text{ else}, \end{cases} \tag{4}
$$

where r_{ω_i} represents the maximum allowable distance of task node ω_i, $\alpha \in [0, 1]$ is a positive constant, and $d(l_{\omega_i}, l_{c_j})$ represent the Euclidean distance between ω_i and c_j. In formula (4), task node ω_i give a preference value of 0 for NCPs whose computing power is less than c_{ω_i} or distance exceeds r_{ω_i}, which means that it will not assign tasks

to such NCPs. The task node ω_i will be used to calculate a higher preference value for NCPs with the distance closer to ω_i and has stronger computing power.

Similarly, a calculation rule of preference list (about NCP ω_i to task node c_j) is defined as follows:

$$f_{c_j \to \omega_i}((l_{\omega_i}, c_{\omega_i}, z_{\omega_i}), (l_{c_j}, c_{c_j}, z_{c_j})) = \begin{cases} 0, & \text{if } c_{\omega_i} \geq c_{c_j} \text{ and } c_{\omega_i} < \chi(c_j), \\ \dfrac{\beta}{d(l_{\omega_i}, l_{c_j})} + \dfrac{1-\beta}{(c_{c_j} - c_{\omega_i})}, & \text{if } \chi(c_j) \leq c_{\omega_i} < c_{c_j}, \end{cases} \tag{5}$$

where $\beta \in [0, 1]$ is a positive constant, $\chi(c_j)$ represents preference of c_j for tasks in terms of computing power, c_j rejects tasks with computing power requirements lower than $\chi(c_j)$, the size of $\chi(c_j)$ can be adjusted according to the size of the task and its own idle resources.

Compared with NCP, task nodes prefer computing nodes with shorter communication distance because they are more sensitive to delay. In order to improve the efficiency of completing the task and reduce the delay, the task node will refuse to interact with the NCP that exceeds the allowable distance. In addition, task nodes prefer NCP with stronger computing power. On the contrary, the NCP prefers the task node which close computing power requirements.

Algorithm 1 Preference list $P(\omega_i)$ of task nodes

Input: $(l_{\omega_i}, l_{\omega_i}, z_{\omega_i})$, $(l_{c_j}, c_{c_j}, z_{c_j})$, r_{ω_i}, formula (4)
Output: Preference list $P(\omega_i)$

1: **for** $i = 1$ to m **do**
2: $j \in \{1, 2, \cdots, n\}$, computing $f_{\omega_i \to c_j}$ based on formula (4)
3: **if** $f_{\omega_i \to c_j} = 0$, **then**
4: $c_j \notin P(\omega_i)$
5: **else**
6: record $f_{\omega_i \to c_j}$
7: **end if**
8: **end for**
9: **while** compare the size of all $f_{\omega_i \to c_j}$ **do**
10: sort by size $P(\omega_i)$
11: **end while**
12: **return** $P(\omega_i)$

Next, we can use computational rules (4) and (5) to calculate the strict preference list corresponding for each node. For all task nodes and NCPs participating in this batch of task allocation, the information between them is shared. The task node can receive all NCPs data information $(l_{c_j}, c_{c_j}, z_{c_j})$, and then calculate the preference list. Specifically, executing Algorithm 1, then the preference lists of task nodes for all

Algorithm 2 Preference list $P(c_j)$ of NCPs

Input: $(l_{\omega_i}, l_{\omega_i}, z_{\omega_i})$, $(l_{c_j}, c_{c_j}, z_{c_j})$, $\chi(c_j)$, and formula (5)
Output: Preference list $P(c_j)$
1: **for** $j = 1$ to n **do**
2: $i \in \{1, 2, \cdots, m\}$, computing $f_{c_j \to \omega_i}$ based on formula (5)
3: **if** $f_{c_j \to \omega_i} = 0$, **then**
4: $\omega_i \notin P(c_j)$
5: **else**
6: record $f_{c_j \to \omega_i}$
7: **end if**
8: **end for**
9: **while** compare the size of all $f_{c_j \to \omega_i}$ **do**
10: sort by size $P(c_j)$
11: **end while**
12: **return** $P(c_j)$

NCPs can be expressed as follows:

$$\begin{aligned} P(\omega_1): \ & c_{v_1^{(1)}} \succ c_{v_2^{(1)}} \succ \cdots \succ c_{v_{\delta_{\omega_1}}^{(1)}}, \\ P(\omega_2): \ & c_{v_1^{(2)}} \succ c_{v_2^{(2)}} \succ \cdots \succ c_{v_{\delta_{\omega_2}}^{(2)}}, \\ & \vdots \\ P(\omega_m): \ & c_{v_1^{(n)}} \succ c_{v_2^{(n)}} \succ \cdots \succ c_{v_{\delta_{\omega m}}^{(n)}}, \end{aligned} \tag{6}$$

where δ_{ω_i} represents the number of NCPs that satisfied the requirements of task node ω_i. For any p and q, if $f_{\omega_i \to c_p} \succ f_{\omega_i \to c_q}$, then $c_{p^{(i)}} \succ c_{q^{(i)}}$ in the preference list of $P(\omega_i)$. For any i, $\{v_1^{(i)}, v_2^{(i)}, \cdots, v_{\delta_{\omega_i}}^{(i)}\} \subset \{1, 2, \cdots, n\}$.

Executing Algorithm 2, the preference lists of NCPs for all task nodes is expressed as follows:

$$\begin{aligned} P(c_1): \ & \omega_{\kappa_1^{(1)}} \succ \omega_{\kappa_2^{(1)}} \succ \cdots \succ \omega_{\kappa_{\sigma_{c_1}}^{(1)}}, \\ P(c_2): \ & \omega_{\kappa_1^{(2)}} \succ \omega_{\kappa_2^{(2)}} \succ \cdots \succ \omega_{\kappa_{\sigma_{c_2}}^{(2)}}, \\ & \vdots \\ P(c_m): \ & \omega_{\kappa_1^{(n)}} \succ \omega_{\kappa_2^{(n)}} \succ \cdots \succ \omega_{\kappa_{\sigma_{c_m}}^{(n)}}, \end{aligned} \tag{7}$$

where σ_{c_j} represents the number of task node that satisfies the requirements of NCP c_j, and $\{\kappa_1^{(j)}, \kappa_2^{(j)}, \cdots, \kappa_{\sigma_{c_j}}^{(j)}\} \subset \{1, 2, \cdots, m\}$, for any $j = 1, 2, \cdots, n$. Note that the preference list (6) and (7) are incomplete, i.e., $\delta_{\omega_i} < n$ and $\sigma_{c_j} < m$.

2.3 Preserving location privacy

Note that the task nodes and NCPs in the model (3) provide actual locations. Obviously, the location information belongs to the user's private information. In fact, neither the task node nor the NCP is willing to provide its actual locations. In order to prevent the leakage of user privacy information, a privacy protection mechanism needs to be established in the task allocation model. Both spatial cloaking and anonymity can be used to protect the user's location privacy information. However, if the adversary possesses specific prior knowledge, the privacy guarantee can be easily lowered, which is called an inference attack [28]. In contrast, the advantage of differential privacy is that it does not require special attack assumptions and does not care about the background knowledge possessed by the attacker. In this paper, we provide a differential privacy protection method for nodes in task allocation.

The basic principle of the Planar Laplace mechanism is to generate obfuscated locations through probability distributions in place of real positions. Given the parameter $\varepsilon \in \mathbb{R}_+$, and the actual location $\tilde{l} \in \mathbb{R}^2$, for any other point $l \in \mathbb{R}^2$, the probability density function of the Planar Laplace mechanism is as follows [16]:

$$M_\varepsilon(l)(\tilde{l}) = \frac{\varepsilon^2}{2\pi} e^{-\varepsilon d(l,\tilde{l})}, \tag{8}$$

where $\frac{\varepsilon^2}{2\pi}$ is a normalization factor, $d(l, \tilde{l})$ is the distance between l and $\tilde{l}$. It is assumed that there are two actual positions l_1 and l_2, and the farthest distance between them is r. Further, it is assumed that the probabilities of these two actual locations producing obfuscated locations l^* by the Planar Laplace mechanism are P_1 and P_2, respectively. Then the Planar Laplace mechanism guarantees that the difference between P_1 and P_2 is at most multiplied by a multiplier $e^{-\varepsilon d(l_1,l_2)}$.

Bordenabe et al. [16] proved that the Planar Laplace mechanism satisfies satisfies the (ε, r)-geo-indistinguishability. Wang et al. [19] proved that the Planar Laplace mechanism satisfies satisfies the ϵ-differential-privacy, ϵ is the privacy budget (the smaller ϵ, the higher privacy). In this paper, we use the the Planar Laplace mechanism to protect the location privacy of task nodes and NCPs.

Under the privacy protection mechanism, the actual location is replaced with the obfuscated locations of the participant, and then the model (3) can be rewritten as follows:

$$\begin{aligned}
&\max J = J_1 + J_2^{-1} + J_3^{-1}, \\
&s.t. \quad \sum_{i=1}^{m} x_{ij} \le z_{c_j}, \quad j = 1, 2, \cdots, n, \\
&\quad \sum_{j=1}^{n} x_{ij} \le z_{\omega_i}, \quad i = 1, 2, \cdots, m, \\
&\quad x_{ij} \in \{0, 1\}, \quad i = 1, 2, \cdots, m, \quad j = 1, 2, \cdots, n, \\
&\quad \text{if } x_{ij} = 1, \text{ then } \omega_i \in P(c_j) \text{ and } c_j \in P(\omega_i),
\end{aligned} \tag{9}$$

where $J_2 = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} d(l^*_{\omega_i}, l^*_{c_j})$, $l^*_{\omega_i}$ is the obfuscated position generated by $l^*_{\omega_i}$ under the Planar Laplace mechanism, and $l^*_{c_j}$ is the obfuscated position generated by l_{c_j} under the Planar Laplace mechanism.

3 Algorithm design and theoretical analysis

3.1 Many-to-many matching algorithm

This task allocation problem (9) is classified as a Mixed Integer Non-Linear Programming (MINLP) problem, which is NP-hard problem. Traditional heuristic algorithms (such as centralized optimization techniques) generate significant overhead and complexity when solving intensive network optimization problems with a huge number of nodes [29]. As a promising task allocation technology, matching theory provides a new method for task allocation between two disjoint sets of heterogeneous environments. Thus, in this paper, we developed a new many-to-many matching algorithm to solve the task allocation model (9). The optimization problem is solved by solving the many-to-many matching problem under incomplete preference list between the task nodes and NCPs.

Aiming at the characteristics of dispersed computing scenarios, there are still some major challenges in solving model (9) with matching algorithms. The number of nodes involved in our task allocation is huge, and each node has explicit additional conditions, which makes the structure of the matching relationship between task nodes and NCPs complicated, and it is impossible to use enumeration or visualization methods for reasoning. Thus, the many-to-many matching algorithm in paper [20] could not be applied to our model. In addition, the preference list between model (9) nodes is incomplete, and the matching relationship has a more complex topology than the literature [30, 31]. Thus, it is necessary to expand on these works and propose a new matching algorithm suitable for our model solving. To express the core idea of many-to-many matching algorithm more clearly, the main matching process in the algorithm 3 is first introduced.

1. In order to improve the efficiency of matching, before performing matching, the structure of the task node's preference list is first optimized to reduce unnecessary matching requests.
2. In the first matching, task nodes sends matching requests to NCPs in sequence according to their preference list, and NCPs will choose to accept or reject according to its preference list.
3. In the second matching, task nodes that have not formed a saturated matching pair continue to send a second matching request to NCPs. The difference is that in the second matching, NCPs need to compare with the first matching pairs when making a decision, and it is possible to break the first match result.
4. Following this method, due to the limited number of nodes and the limited length of the preference list, the algorithm will definitely end after a limited number of matching.

The specific formal algorithm steps are as follows.

Step 1: Update preference list (6). Note that this situation exists in the preference list (6) and (7): $c_j \in P(\omega_i)$, but $\omega_i \notin P(c_j)$. In this situation, ω_i is not necessary to send a matching request to c_j, which can also improve the efficiency of the algorithm. Because task nodes cannot obtain the preference list of NCPs through direct observation, so they must interact with each other to achieve the purpose of updating the preference list.

Specifically, for each $\omega_i \in T$, sending a matching request to each NCP in set $P(\omega_i)$, the NCP $c_j \in P(\omega_i)$ only needs to give back 'Yes' (when $\omega_i \in P(c_j)$) or 'No' (when $\omega_i \notin P(c_j)$). When ω_i receives c_j's feedback message as 'No', ω_i removes c_j from its preference list. Perform this step for all elements in T, and get the following new preference list

$$\begin{aligned} P(\omega_1):&\ c_{\nu_1^{(1)}} \succ c_{\nu_2^{(1)}} \succ \cdots \succ c_{\nu_{\rho_1}^{(1)}}, \\ P(\omega_2):&\ c_{\nu_1^{(2)}} \succ c_{\nu_2^{(2)}} \succ \cdots \succ c_{\nu_{\rho_{\omega_2}}^{(2)}}, \\ &\ \vdots \\ P(\omega_m):&\ c_{\nu_1^{(n)}} \succ c_{\nu_2^{(n)}} \succ \cdots \succ c_{\nu_{\rho_{\omega m}}^{(n)}}. \end{aligned} \tag{10}$$

Obviously, $\rho_{\omega_i} \leq \delta_{\omega_i}$.

Step 2: Divide all task nodes into two categories according to the number of task splitting, the task nodes with one job denoted by $\{\omega_{p_1}, \omega_{p_2}, \cdots, \omega_{p_l}\}$, and task nodes with many job denoted by $\{\omega_{q_1}, \omega_{q_2}, \cdots, \omega_{q_k}\}$, where $z_{\omega_{p_i}} = 1$, $(i = 1, 2, \cdots, l)$, and $z_{\omega_{q_j}} > 1$, $(j = 1, 2, \cdots, k)$, $l + k = m$. Make all task nodes send a matching request to the NCPs for the first time based on the preference list as formulas (11) and (12),

$$\begin{pmatrix} \omega_{p_1} & \omega_{p_2} & \cdots & \omega_{p_l} \\ \downarrow & \downarrow & & \downarrow \\ c_{\nu_1^{(p_1)}} & c_{\nu_1^{(p_2)}} & \cdots & c_{\nu_1^{(p_l)}} \end{pmatrix}, \tag{11}$$

$$\begin{pmatrix} \omega_{q_1} & \omega_{q_2} & \cdots & \omega_{q_k} \\ \downarrow & \downarrow & & \downarrow \\ (c_{\nu_1^{(q_1)}}, \cdots, c_{\nu_{z_{\omega q_1}}^{(q_1)}}) & (c_{\nu_1^{(q_2)}}, \cdots, c_{\nu_{z_{\omega q_2}}^{(q_2)}}) & \cdots & (c_{\nu_1^{(q_k)}}, \cdots, c_{\nu_{z_{\omega q_k}}^{(q_k)}}) \end{pmatrix}, \tag{12}$$

where $\nu_j^{(i)}$ indicates that the position of NCP $c_{\nu_j^{(i)}}$ in the preference list of task node ω_i is j-th. Note that the matching request in the formula (12) is only a special case, which means that

$$z_{c_{\nu_1^{(q_i)}}} = z_{c_{\nu_1^{(q_i)}}} = \cdots = z_{c_{\nu_{z_{\omega q_i}}^{(q_i)}}} = 1. \tag{13}$$

If the Eq. (13) does not hold, it can be further discussed based on the relationship between $z_{\omega_{q_i}}$ and $z_{c_{v_1^{(q_i)}}}, z_{c_{v_2^{(q_i)}}}, \cdots, z_{c_{v_{z_{\omega_{q_i}}}^{(q_i)}}}$. If $z_{\omega_{q_i}} \leq z_{c_{v_1^{(q_i)}}}$, then the matching request has the following form (take ω_{q_i} as an example)

$$\left(\begin{array}{c} \omega_{q_1} \\ \downarrow \\ \underbrace{c_{v_1^{(q_i)}}, \cdots, c_{v_1^{(q_i)}}}_{z_{\omega_{q_i}} th} \end{array} \right), \tag{14}$$

where

$$\underbrace{c_{v_1^{(q_i)}}, \cdots, c_{v_1^{(q_i)}}}_{z_{\omega_{q_i}} th}$$

means there are $z_{\omega_{q_i}}$ NCPs. If $z_{c_{v_1^{(q_i)}}} < z_{\omega_{q_i}} \leq (z_{c_{v_1^{(q_i)}}} + z_{c_{v_2^{(q_i)}}})$, then the matching request has the following form

$$\left(\begin{array}{c} \omega_{q_1} \\ \downarrow \\ \underbrace{c_{v_1^{(q_i)}}, \cdots, c_{v_1^{(q_i)}}}_{z_{c_{v_1^{(q_i)}}} th}, \underbrace{c_{v_2^{(q_i)}}, \cdots, c_{v_2^{(q_i)}}}_{\left(z_{\omega_{q_i}} - z_{c_{v_1^{(q_i)}}}\right) th} \end{array} \right). \tag{15}$$

Otherwise, we need to continue to discuss the relationship between $z_{\omega_{q_i}}$ and $(z_{c_{v_1^{(q_i)}}} + z_{c_{v_2^{(q_i)}}} + z_{c_{v_3^{(q_i)}}})$. For the convenience of discussion, we describe the matching request as formula (12) when it does not affect the design of the subsequent algorithm.

Step 3: Record the NCPs in the matching requests (11) and (12) as the set C_1. For any $c_i \in C_1$, we assume that $K(c_i)$ represents the number of occurrences of c_i in the request (11) and (12). If $K(c_i) \leq z_{c_i}$, then all matching requests about c_i form matching pair. If $K(c_i) > z_{c_i}$, all requesters ($K(c_i)$ task nodes) have a sequence in preference list of c_i, and c_i selects z_{c_i} requesters with higher priority to form matching pair. Specifically, in this case, we assume that the preference list of c_i has the following form

$$P(c_i) = \omega_{\kappa_1^{(i)}} \succ \omega_{\kappa_2^{(i)}} \succ \cdots \succ \omega_{\kappa_{\sigma_{c_i}}^{(i)}}. \tag{16}$$

Defining a mapping $\tau_{c_i}(\omega_j) = \mu$, its meaning is that the μ-th preference of c_i is the task node ω_j. We record the $K(c_i)$ task nodes that who sent the matching request to the NCP c_i in the matching request (11) and (12), are $\omega_{\lambda_1}, \omega_{\lambda_2}, \cdots$, and $\omega_{\lambda_{K(c_i)}}$, respectively. Arrange $\tau_{c_i}(\omega_{\lambda_1}), \tau_{c_i}(\omega_{\lambda_2}), \cdots$, and $\tau_{c_i}(\omega_{\lambda_{K(c_i)}})$ in ascending lists, and take the top z_{c_i}. If $K(c_i) \leq z(c_i)$ is satisfied for all $c_i \in C_1$, then the algorithm ends

and all matching pairs are output. Otherwise, there is $c_i \in C$ such that $K(c_i) > z_{c_i}$. From **Step 3**, there is $K(c_i) - z_{c_i}$ matching requests was rejected.

Step 4: For task nodes that do not form a matching pair (i.e., in (11)) and task nodes that do not form a saturated matching pair (i.e., in (12)) are recorded as a set Ω_1. For any $\omega_i \in \Omega_1$, we assume that the preference list of NCP that form a matching pair with ω_i is: $c_{v_1}^{(i)} \succ c_{v_2}^{(i)} \succ \cdots \succ c_{v_\gamma}^{(i)}$, (where $\gamma < z_{\omega_i}$). We need to investigate whether $c_{v_\gamma}^{(i)}$ has formed a saturated matching pair, i.e., compareing the size of $K(c_{v_\gamma}^{(i)})$ and $z_{c_{v_\gamma}^{(i)}}$. If $K(c_{v_\gamma}^{(i)}) < z_{c_{v_\gamma}^{(i)}}$, then ω_i needs to $c_{v_\gamma}^{(i)}$ and NCP whose preference is lower than $c_{v_\gamma}^{(i)}$ send $z_{\omega_i} - \gamma$ matching requests. If $K(c_{v_\gamma}^{(i)}) = z_{c_{v_\gamma}^{(i)}}$, then ω_i needs to NCP whose preference is lower than $c_{v_\gamma}^{(i)}$ send $z_{\omega_i} - \gamma$ matching requests.

According to the above method, a matching request is sent to all elements in Ω_1 for the second time, then returns to **Step 2**. For task nodes that do not form a matching pair or do not form a saturated matching pair, they are still recorded as a set Ω_1.

Step 5: Repeat **Step 4** and **Step 3** for the set Ω_1 until $\Omega_1 = \emptyset$, i.e., all elements in Ω_1 form a saturated matching pair, or a task node has sent a matching request to all NCP in its preference list, but still has not form a saturated matching pair. The algorithm is terminated and all formed pairs are output. The many-to-many matching algorithm is outlined in Algorithm 3.

In order to clearly illustrate the implementation of the algorithm, an example is taken as follows. **Example** The task node set is denoted as

$$\Omega = \{(\omega_1, 1), (\omega_2, 3), (\omega_3, 3), (\omega_4, 5), (\omega_5, 1)\}, \tag{17}$$

The second coordinate in (ω_i, x) indicates that the task of task node ω_i can be divided into x jobs. The NCP set is denoted as

$$C = \{(c_1, 2), (c_2, 2), (c_3, 1), (c_4, 2), (c_5, 2), (c_6, 5), (c_7, 1), (c_8, 1)\}. \tag{18}$$

The second coordinate in (c_j, y) indicates that the NCP c_j can complete y jobs. The preference of task nodes to NCP is as follows:

$$\begin{aligned} P(\omega_1) =& c_5 \succ c_3 \succ c_2, \quad P(\omega_2) = c_2 \succ c_5 \succ c_7 \succ c_1, \\ P(\omega_3) =& c_5 \succ c_6 \succ c_2 \succ c_4 \succ c_3, \quad P(\omega_4) = c_5 \succ c_6 \succ c_7 \succ c_8 \succ c_3 \succ c_4, \\ P(\omega_5) =& c_4 \succ c_3 \succ c_1 \succ c_2 \succ c_5 \succ c_8 \succ c_7 \succ c_6, \end{aligned} \tag{19}$$

The preference of NCP to task nodes is as follows:

$$\begin{aligned} P(c_1) =& \omega_3 \succ \omega_2 \succ \omega_1 \succ \omega_4, \quad P(c_2) = \omega_3 \succ \omega_4 \succ \omega_1 \succ \omega_2, \\ P(c_3) =& \omega_2 \succ \omega_3 \succ \omega_1 \succ \omega_5, \quad P(c_4) = \omega_5 \succ \omega_4 \succ \omega_1 \succ \omega_3, \\ P(c_5) =& \omega_5 \succ \omega_4 \succ \omega_2 \succ \omega_1 \succ \omega_3, \quad P(c_6) = \omega_1 \succ \omega_2 \succ \omega_3 \succ \omega_5, \\ P(c_7) =& \omega_1 \succ \omega_4 \succ \omega_3 \succ \omega_2, \quad P(c_8) = \omega_5 \succ \omega_3 \succ \omega_1. \end{aligned} \tag{20}$$

Algorithm 3 Many-to-many matching

Input: T, C, $P(\omega_i)$, $P(c_j)$, z_{ω_i}, z_{c_j}, where $i \in [1, m]$ and $j \in [1, n]$
Output: All matching pairs $\omega_i \leftrightarrow c_j$

for $i \leq m$ **do**
Update $P(\omega_i)$
  if $\omega_i \in P(c_j)$, where $c_j \in P(\omega_i)$, **then**
  keep c_j in the $P(\omega_i)$
  else
  remove c_j from the $P(\omega_i)$
  end if
end for
while For all $\omega_i \in \Omega$ send matching requests based on (10), and get (11) and (12) **do**
  if $K(c_j) \leq z_{c_j}$, **then**
  c_j accept all matching requests
  else
  c_j selects the top z_{c_j} task nodes and forms a pair with them
  end if
end while
while Build a set Ω_1, for all $\omega_i \in \Omega_1$, send matching requests based on (10)
  if $K(c_j) \leq z_{c_j}$, where $c_j \in C_1$, **then**
c_j accept all matching requests
  else
c_j selects the top z_{c_j} task nodes and forms a pair with them,
  end if
end while
while Update Ω_1 **do**
  if $\Omega_1 = \emptyset$, **then**
  output all matching pairs, the algorithm ends
Else
  execution algorithm **16-22**
  end if
end while
return $\omega_i \leftrightarrow c_j$

Next, we apply the many-to-many matching algorithm proposed in this paper to solve the above task allocation problem.

Step 1 Update preference list (19). The node ω_1 sends matching requests to c_5, c_3, and c_2, respectively, and gets feedback all is 'Yes', then keep c_5, c_3, and c_2 in the preference list of ω_1. Perform this step on all task nodes to get a new preference list as follows:

$$\begin{aligned} P(\omega_1) =& c_5 \succ c_3 \succ c_2, \quad P(\omega_2) = c_2 \succ c_5 \succ c_7 \succ c_1, \\ P(\omega_3) =& c_5 \succ c_6 \succ c_2 \succ c_4 \succ c_3, \quad P(\omega_4) = c_5 \succ c_7 \succ c_4, \\ P(\omega_5) =& c_4 \succ c_3 \succ c_5 \succ c_6. \end{aligned} \tag{21}$$

Step 2 The first matching request is as follows:

$$\begin{pmatrix} \omega_1 & \omega_5 \\ \downarrow & \downarrow \\ c_5 & c_4 \end{pmatrix} \begin{pmatrix} \omega_2 & \omega_3 & \omega_4 \\ \downarrow & \downarrow & \downarrow \\ (c_2, c_2, c_5) & (c_5, c_5, c_6) & (c_5, c_5, c_7, c_4, c_4) \end{pmatrix}. \tag{22}$$

Step 3 Since $K(c_2) \leq z_{c_2}$, $K(c_6) \leq z_{c_6}$, and $K(c_7) \leq z_{c_7}$, it can initially form a matching pair as follows:

$$\begin{pmatrix} \omega_2 & \omega_3 & \omega_4 \\ \updownarrow & \updownarrow & \updownarrow \\ (c_2, c_2, *) & (*, *, c_6) & (*, *, c_7, *, *) \end{pmatrix}. \tag{23}$$

For $K(c_5) > z_{c_5}$, we need to find out the task nodes that send matching requests to c_5, and then compare these task nodes in c_5. In the type (22) there are ω_2, ω_3, and ω_4 sent a matching request to c_5. Note that the preference list of c_5 are

$$P(c_5): \ \omega_5 \succ \omega_4 \succ \omega_2 \succ \omega_1 \succ \omega_3, \tag{24}$$

and $z_{c_5} = 2$, so c_5 will reject ω_1, ω_2, ω_3 and accept ω_4, forming two matching pairs. Similarly, c_4 will accept ω_5 and reject a matching requests ω_4, forming two matching pairs. In summary, the matching pair formed is as follows:

$$\begin{pmatrix} \omega_1 & \omega_5 \\ \updownarrow & \updownarrow \\ * & c_4 \end{pmatrix} \begin{pmatrix} \omega_2 & \omega_3 & \omega_4 \\ \updownarrow & \updownarrow & \updownarrow \\ (c_2, c_2, *) & (*, *, c_6) & (c_5, c_5, c_7, c_4, *) \end{pmatrix} \tag{25}$$

Step 4 Note that neither of $\omega_1, \omega_2, \omega_3$, and ω_4 all form saturated matching pairs, i.e., $\Omega_1 = \{\omega_1, \omega_2, \omega_3, \omega_4\}$. The last request in ω_2 was rejected by c_5, so ω_2 sends requests to NCP whose preference list is lower than c_5, i.e., $\omega_2 \rightarrow c_7$. In the matching pairs formed by ω_3, the last term NCP c_6 has not reached saturation. Therefore, ω_3 sends requests to c_6 and NCP whose preference is lower than c_6, i.e., $\omega_3 \rightarrow c_6$. Since the last request in ω_4 was rejected by c_4, there is no more preferred NCP in the preference list of ω_4. The task of ω_4 can only be temporarily put on hold and is unwilling to send a matching request anymore. The second matching request is described as

$$\begin{pmatrix} \omega_1 & \omega_2 & \omega_3 \\ \downarrow & \downarrow & \downarrow \\ c_3 & c_7 & (c_6, c_6) \end{pmatrix}. \tag{26}$$

In matching request (26), note that $K(c_7) = 2 > z_{c_7}$, c_7 will reject ω_2, and accept ω_4 to form a matching pairs because c_7 prefers ω_4 to ω_2. c_6 did not reach saturation, and formed two matching pairs with ω_3, i.e., $\omega_3 \leftrightarrow c_6$. NCP c_3 appears for the first time and forms a matching pair with ω_1, i.e., $\omega_1 \leftrightarrow c_3$. Update matching pair (25), we have

$$\begin{pmatrix} \omega_1 & \omega_5 \\ \updownarrow & \updownarrow \\ c_3 & c_4 \end{pmatrix} \begin{pmatrix} \omega_2 & \omega_3 & \omega_4 \\ \updownarrow & \updownarrow & \updownarrow \\ (c_2, c_2, *) & (c_6, c_6, c_6) & (c_5, c_5, c_7, c_4, *) \end{pmatrix}. \tag{27}$$

Updating set Ω_1, we can get $\Omega_1 = \{\omega_2\}$.

Step 5 Similar to the discussion of **Step 4**, ω_2 send the third matching request as $\omega_2 \rightarrow c_1$. Since c_1 appears for the first time, it can form a matching pair with ω_2, i.e.,

$\omega_2 \leftrightarrow c_1$. Node that ω_4 sends two matching requests to c_4, and c_4 only accepted one and formed a matching pair, i.e., $\omega_4 \leftrightarrow c_4$. Updating matching pair (27), we get the latest matching pair

$$\begin{pmatrix} \omega_1 & \omega_5 \\ \updownarrow & \updownarrow \\ c_3 & c_4 \end{pmatrix} \begin{pmatrix} \omega_2 & \omega_3 & \omega_4 \\ \updownarrow & \updownarrow & \updownarrow \\ (c_2, c_2, c_1) & (c_6, c_6, c_6) & (c_5, c_5, c_7, c_4, *) \end{pmatrix}. \tag{28}$$

Although ω_4 does not form a saturated matching pair, there is no NCP that can send matching requests. Therefore, $\Omega_1 = \emptyset$ can be obtained, the algorithm ends, and outputs matching pair (28).

Note that in the final matching pair (28), there is still a job in the task node ω_4 that has not been allocated successfully. In order to describe the completion efficiency of the matching algorithm, we need to calculate the task completion rate. In view of the definition (1), we have

$$F(\mu) = \frac{\sum_{i=1}^{5} z'_{\omega_i}}{\sum_{i=1}^{5} z_{\omega_i}} = \frac{12}{13} = 92.31\%. \tag{29}$$

For the above example, the task nodes initiate a total of three matching requests and formed three matching pairs in the example, respectively, i.e., (23), (26), and the latest matching pair (27). Therefore, the number of iterations is $G(\mu) = 3$. To evaluate the performance of a matching μ, the task completion rate $F(\mu)$ and the number of iterations $G(\mu)$ are two important indicators.

3.2 Theoretical analysis

In this subsection, we mainly discuss the stability and time complexity of the many-to-many matching Algorithm 3. According to the definition in Ref. [25]: a matching μ is stable if it is not blocked by any individual or any task-NCP pair; a stable matching is called optimal if every test is at least as well off as under any other matching. For the NP-hard problems, we weaken the requirement objective, do not search for the global optimal solution, but only search for the unilateral optimal (about task nodes). In the following theorem, we give the conclusion about the unilateral optimal solution of the task node.

Theorem 1 *For the matching problem* $(\Omega, C, P_{\Omega\to C}, P_{C\to\Omega}, z_\Omega, z_C)$*, if the preference list (6) and (7) are strict, then the matching obtained by applying the Many-to-many matching Algorithm 3 must be unique, stable, and optimal matching for the task node. That is, the corresponding optimization model (9) to the matching problem* $(\Omega, C, P_{\Omega\to C}, P_{C\to\Omega}, z_\Omega, z_C)$ *admits a unique, stable, and optimal solution for the task node.*

Proof The existence of solution is proved firstly. Note that the number of tasks Ω and NCP C is limited, so the preference list (6) for each task must be limited. In the rules of our algorithm, if the task ω_i is rejected by NCP c_j, it is not allowed to send matching

requests repeatedly, and can only send requests to the NCP with lower preference list. Therefore, it will be terminate the algorithm after a limited number of requests.

Next, the stability of matching μ is proved. If the matching is unstable, then there are two matching pairs $\omega_i \leftrightarrow c_j$ and $\omega_i^{'} \leftrightarrow c_j^{'}$, although ω_i prefers $c_j^{'}$ to c_j and $c_j^{'}$ prefers ω_i to $\omega_i^{'}$, but $c_j^{'} \notin \mu(\omega_i)$. Since ω_i prefers $c_j^{'}$ to c_j, ω_i must send a matching request to $c_j^{'}$ earlier than c_j. In view of $c_j^{'}$ prefers ω_i to $\omega_i^{'}$ and $\omega_i^{'} \leftrightarrow c_j^{'}$, therefore, $c_j^{'}$ will inevitably accept ω_i's matching request, a matching pair $\omega_i \leftrightarrow c_j^{'}$ is formed. Note that the condition $c_j^{'} \notin \mu(\omega_i)$, this is contradiction. Therefore, there is no such unstable matching pair.

Finally, it is suggested that the matching result must be optimal for task nodes. Each task node selects the favorite among all NCPs (in her preference list) that send matching requests to it and forms a matching pair. For any $\omega_i \in \Omega$, the algorithm can guarantee the following two properties

1) For any $c_j \in P(\omega_i)$, then ω_i prefers c_j to $c_{j'}$, where $c_{j'} \in \Omega$ and $c_{j'} \notin P(\omega_i)$. This is obvious, because the preference list $P(\omega_i)$ is calculated according to your own preference.
2) For any $c_{j'} \in \mu(\omega_i)$, then ω_i prefers $c_{j'}$ to $c_{j''}$, where $c_{j''} \in P(\omega_i)$ and $c_{j''} \notin \mu(\omega_i)$. If there is $c_{j*} \in P(\omega_i)$ and ω_i prefers c_{j*} to $c_{j'}$, but $c_{j*} \notin \mu(\omega_i)$, then there is only one situation that ω_i does not appear in preference list c_{j*}. According to the preference list update rule in Algorithm 3, ω_i has no right to send a matching request to c_{j*}. Otherwise, ω_i and c_{j*} will form a matching pair, which contradicts $c_{j*} \notin \mu(\omega_i)$.

According to the arbitrariness of c_j and $c_{j'}$, the desired conclusion can be obtained.

Remark 1 From the property $i)$ and property $ii)$ in the proof of the Theorem 1, it can be known that the so-called task optimal is not an absolute optimal solution. Due to the matching request sent by the task node, it has an advantage over NCPs in the matching process. Of course, the algorithm also guarantees that NCP selects the best among all available task nodes.

Theorem 2 *The total time complexity of the many-to-many matching Algorithm 3 is polynomial in the order of* $\mathcal{O}(n^2 * m)$.

Proof During the first matching process (steps 9-15), m nodes send matching requests, and the list length of each node does not exceed n, so the algorithm has an $\mathcal{O}(n * m)$ worst-case time bound. For any task node ω_i, it is rejected at most n times, that is, at most n matches are performed (i.e., steps 16-22 at most cycles $(n-1)$ times). In steps 1-8, the list is updated so that the preference list of each task node becomes shorter, which not only does not increase the total computational complexity, but optimizes the computational structure. Thus, the many-to-many matching Algorithm 3 has an $\mathcal{O}(n^2 * m)$.

4 Numerical simulation

4.1 Simulation setup

The experimental data includes information the label of the node (i.e., ω_i or c_j), the coordinate position of the node (i.e., $(x_{\omega_i}, y_{\omega_i})$ or (x_{c_j}, y_{c_j})), the size of the task (z_{ω_i} or z_{c_j}), the additional requirements of the task node (including minimum computing power requirement c_{ω_i} and allowable distance r_{ω_i}), the physical performance of the NCP c_{c_j} and its requirements $\chi(c_j)$ for task screening (i.e., threshold).

Data set construction: The format of the data set we constructed is in Table 1 and Table 2, respectively. Specifically, two types of nodes are generated from randomly in the area $1000 * 1000$, and then the coordinates of nodes are recored(the number of nodes depends on the experimental requirements). Specifically, two types of nodes are generated from certain construction methods in the area $1000 * 1000$, and then the coordinates of nodes are recored(the number of nodes depends on the experimental requirements). For example, in a 1000*1000 area, there is a 50*50 dense area where 200 points are randomly generated, and 10 points are randomly generated in the remaining area. By this method, the globally sparse and locally clustered node distribution can be generated. As long as the node coordinates are determined, we can always calculate the preference list for all nodes. For all z_{ω_i} and z_{c_j}, integers are randomly selected in interval [1, 5], and for all (c_{ω_i} and c_{c_j}), integers are randomly selected in interval [1, 5]. As for our choice, $r_{\omega_i} = 8$, (for all i), $\chi(c_j) = 0.5c_{c_j}$, $\alpha = 0.7$, and $\beta = 0.4$.

Calculate preference list and match tasks: the data is input in Table 1 as Algorithm 1 and the algorithm is executed to get the preference list of each task node about all NCPs. Similarly, the data is input in Table 2 as Algorithm 2 and the algorithm is executed to get the preference list of each NCP about all tesk nodes.

Finally, the obtained data set is input into Algorithm 3 to get our matching result. Python has been used for validating the proposed approach experimentally. The input of the program includes: data sets Table 1, Table 2, the output of the Algorithm 1 and Algorithm 2. The output of the algorithm is a successful matching pair.

Next, two types of experiments are set up. In the first experiment, it mainly verifies that the solution obtained by the many-to-many matching algorithm proposed in this paper is stable and optimal for task nodes. In the second experiment, the performance of the many-to-many matching algorithm is verified by setting different comparative experiments.

Table 1 Distribution information of task nodes

Tags	Location	Computing power	Jobs	Threshold
ω_1	$(x_{\omega_1}, y_{\omega_1})$	c_{ω_1}	z_{ω_1}	r_{ω_1}
ω_2	$(x_{\omega_2}, y_{\omega_2})$	c_{ω_2}	z_{ω_2}	r_{ω_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ω_m	$(x_{\omega_m}, y_{\omega_m})$	c_{ω_m}	z_{ω_m}	r_{ω_m}

Table 2 Distribution information of NCPs

Tags	Location	Computing power	Jobs	Threshold
c_1	(x_{c_1}, y_{c_1})	c_{c_1}	z_{c_1}	$\chi(c_1)$
c_2	(x_{c_2}, y_{c_2})	c_{c_2}	z_{c_2}	$\chi(c_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_n	(x_{c_n}, y_{c_n})	c_{c_n}	z_{c_n}	$\chi(c_n)$

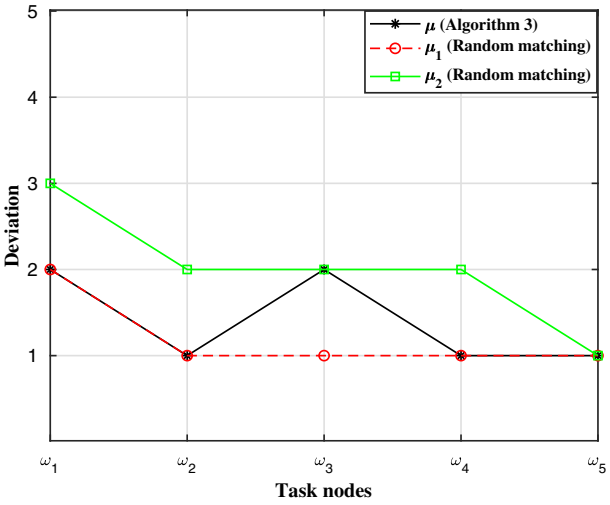

Fig. 2 Degree of deviation in different matching methods

4.2 Results and discussions

In order to verify the conclusion of Theorem 1, the degree of deviation is introduced as a measure of the matching result. Assuming that c_j has the highest priority among the successfully matched results of ω_i, the position of c_j in $P(\omega_i)$ is called the deviation degree of ω_i. Obviously, the smaller the deviation is, the higher the satisfaction of the task node with the matching conclusion will be.

As **Example** 3.1 in Sect. 3 is taken as an illustration, and the deviation degree of task nodes is shown by the black line in Fig. 2. The blue line and the red line respectively indicate the degree of deviation under the two random matches, and $y = 1$ represents the lowest degree of deviation. Since the degree of deviation of μ_1 is lower than that of Algorithm 3 in this paper, the stability of μ_1 needs to be verified.

In μ_1, ω_3 forms a matching pair with at least c_5, which will inevitably destroy the matching pair $\omega_4 \leftrightarrow c_5$. However, c_5 prefers ω_4 to ω_3, and ω_4 is the most preferred position in $P(c_5)$. There is at least one unstable matching in μ_1, which means μ_1 is an unstable matching. It can be further verified that the degrees of deviation lower than Algorithm 3 are all unstable matching. This verifies the conclusion of Theorem 1 that the solution obtained by Algorithm 3 is stable and optimal for task nodes.

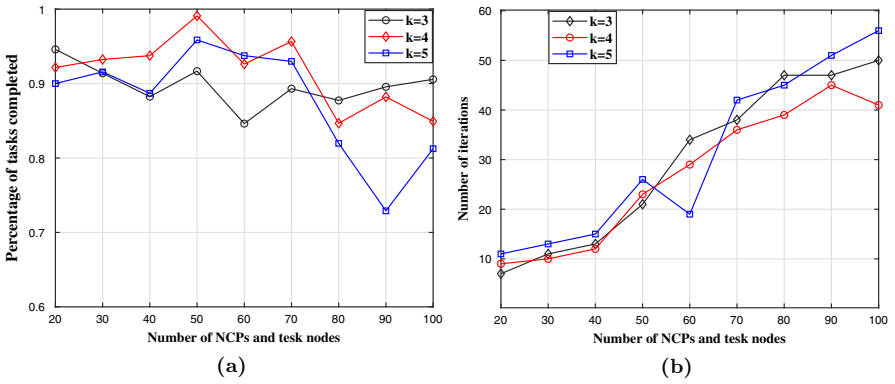

Fig. 3 Task completion rate and number of iterations of the **Example 1**

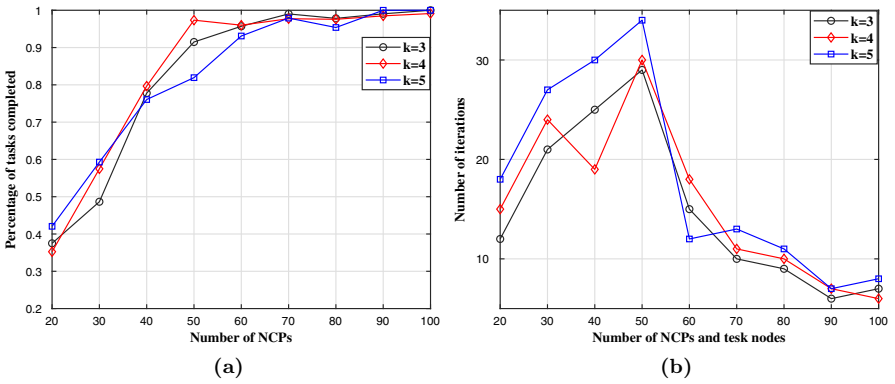

Fig. 4 Task completion rate and number of iterations of the **Example 2**

In more detail, among the three optimization objectives, minimizing response time and communication distance are achieved in accordance with certain preference rules during the matching process. Therefore, we only need to verify the task completion rate of the model to test the optimality of the solution obtained in the Algorithm 3. Next, two example are given to examine the task completion rate of the model.

Example 1 The task nodes and NCPs have the same number, which are increased from 20 to 100, respectively. An important parameter introduced to replace z_{ω_i} and z_{c_j}, i.e., set $k = \max_{1\le i\le m, 1\le j\le n}\{z_{\omega_i}, z_{c_j}\}$, where k represents another topological property of the network in addition to the number and location of nodes, which is called the maximum task amount of nodes. The task completion rate is shown in Fig. 3(a), and the number of iterations required for the matching process is shown in Fig. 3(b).

Note that when the number of nodes is equal, the task completion rate for each matching is more than 70%, concentrated around 90%, and it does not show regularity with the change in the number of nodes. The number of iterations increases as the number of nodes increases, and as k increases, it also increases. However, it should be noted that the increase mentioned here is not strictly a monotonic increase, but a

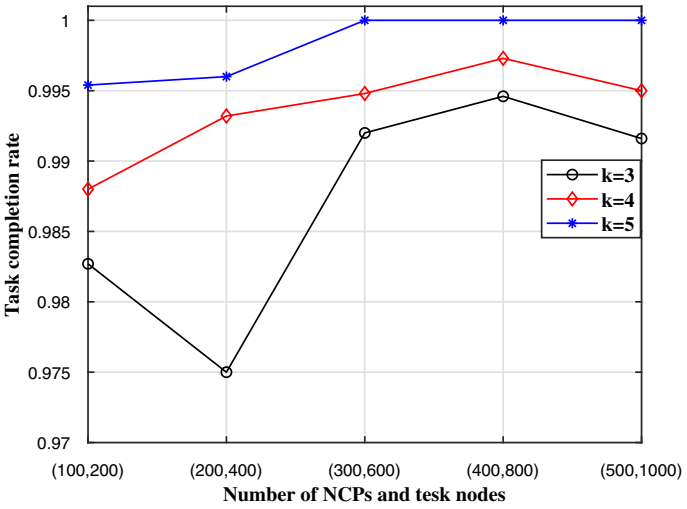

Fig. 5 Task completion rate when the ratio of task node to NCP is 1: 2

description of a general trend, and individual inconsistencies are allowed. For example, in terms of the number of nodes, 60 nodes (details such as (60 60 5)) are significantly more than 50 (details such as (50 50 5)) nodes, but the number of iterations is smaller than the latter. An important reason for this is that in addition to the number of nodes, the key factors that determine the difficulty of matching are related to the network topology of the nodes, which can be uniquely reflected in the preference list.

Example 2 The number of task nodes is fixed at 50, and the number of NCPs is increased from 20 to 100. The task completion rate is shown in Fig. 4(*a*), and the number of iterations required for the matching process is shown in Fig. 4(*b*). Obviously, as the number of NCPs increases, the task completion rate shows an increasing trend. In particular, when the number of NCPs exceeds 60, the task completion rate can reach more than 90%, and when the number of NCPs reaches 80, the task completion rate can reach 98.70%. This experimental group shows that an effective way to improve the completion rate of each matching task is to increase the proportion of NCPs. When the ratio of NCPs to task node reaches $50 : 80 = 1 : 1.6$, the task completion rate can reach 98.70%.

In order to get a more general and stable conclusion, five sets of data are established, the ratio of task node to NCP in each set of data is 1:2, and the task completion rate is shown in Fig. 5. After calculation, the task completion rate can reach an average of 99.33%.

Since the arrival of task nodes and NCPs obey a certain random process, so the arrival rate can be controlled. It is only necessary to adjust the intensities to control the ratio of task nodes to NCPs to achieve a higher task completion rate. An interesting conclusion is that as the task completion rate increases, the number of iterations decreases significantly, even if the number of nodes increases. In short, by increasing

the ratio of NCPs to task nodes, the task completion rate can not only be improved, but also the number of iterations can be reduced (i.e., increased efficiency).

Compared with the existing work [2, 10], the innovation of our work is that we consider the heterogeneity of nodes and the difference of tasks, which are the most important features of task allocation in dispersed computing scenarios. In terms of performance indicators, Ref. [2, 10] mainly aims at maximizing network throughput, while the model in this paper mainly aims to optimize the task completion rate. In other scenarios, such as business process scenarios, the author also uses the task completion rate as the research goal, and the conclusion reached is that the completion rate is always around 85%, and the highest is around 90%. The task completion rate of this paper can be stabilized at 99.33% when the ratio of task nodes to NCPs reaches 1:2. Obviously, the conclusion of this paper is improved on the basis of Ref. [32].

5 Conclusion

In this paper, a task allocation model is studied in dispersed computing environment, where the task node publish tasks as well as NCPs complete tasks in a collaborative way. In the model, we aims at with maximizing task completion rate, minimizing response time, and minimizing communication distance. Specifically, we first introduced a incomplete preference list to quantitatively characterize complex requirements between task nodes and NCPs. These requirements are usually difficult to formally reflect, and the preference list just solves this problem. Secondly, we developed a multi-objective optimization model is established based on an incomplete preference list. In fact, the model can be transformed into a matching problem between two groups (task nodes and NCPs). We then proposed a new many-to-many matching algorithm, and obtained the stability and the unilateral optimal solution (for task nodes) of the model. In addition, the location privacy-preserving of the node is realized by applying the planar Laplace mechanism. Finally, we validated the performance of the task allocation model and many-to-many matching algorithm in different scenarios through theoretical proof as well as numerical simulation.

Acknowledgements This work was supported in part by the National Science Foundation Project of P. R. China (No. 61931001), the Scientific and Technological Innovation Foundation of Foshan, USTB (No. BK20AF003), NSF under Grants IIS-1838024, CNS-1950485, and OIA-2148788.

Declarations

Conflict of interest We claim that we have no conflict of interest with other researchers with regard to the paper.

References

1. Schurgot MR, Wang M, Conway AE, Greenwald LG, Lebling PD (2019) A dispersed computing architecture for resource-centric computation and communication. IEEE Commun Mag 57(7):13–19
2. Yang CS, Pedarsani R, Avestimehr AS (2019) Communication-aware scheduling of serial tasks for dispersed computing. IEEE ACM Trans Network 27(4):1330–1343

3. Knezevic A, Nguyen Q, Tran JA, Ghosh P, Annavaram M (2017) CIRCE-a runtime scheduler for DAG-based dispersed computing. In: Proceedings of the 2nd ACM/IEEE symposium on edge computing, San Jose/Silicon Valley, CA, USA, pp 1–2
4. Conway AE, Wang M, Ljuca E, Lebling PD (2020) A Dynamic Transport Overlay System for Mission-Oriented Dispersed Computing Over IoBT. In: MILCOM 2019–2019 IEEE military communications conference (MILCOM), Norfolk, VA, USA, pp 815–820
5. Yang H, Li G, Sun G, et al (2021) Dispersed computing for tactical edge in future wars: vision, architecture, and challenges. Wirel Commun Mob Comput 2021:8899186:1-8899186:31
6. Kao YH, Krishnamachari B, Ra MR, Bai F (2017) Hermes: latency optimal task assignment for resource-constrained mobile computing. IEEE T Mobile Comput 16(11):3056–3069
7. Wang H, Gong J, Zhuang Y, Shen H, Lach J (2017) Healthedge: task scheduling for edge computing with health emergency and human behavior consideration in smart homes. In: 2017 IEEE international conference on big data (big data), MA, USA, pp 1213–1222
8. Gu Y, Wu CQ, Liu X, Yu D (2013) Distributed throughput optimization for large-scale scientific workflows under fault-tolerance constraint. J Grid Comput 11(3):361–379
9. García-Valls M, Dubey A, Botti V (2018) Introducing the new paradigm of social dispersed computing: applications, technologies and challenges. J Supercomput 91:83–102
10. Hu D, Krishnamachari B (2019) Throughput optimized scheduler for dispersed computing systems. In: 7th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud), Newark, CA, USA, pp 76–84
11. Zhou C, Gong C, Hui H, Lin F, Zeng G (2021) A task-resource joint management model with intelligent control for mission-aware dispersed computing. China Commun 18(10):214–232
12. Ghosh P, Nguyen Q, Krishnamachari B (2019) Container orchestration for dispersed computing, In Proceedings of the 5th International Workshop on Container Technologies and Container Clouds, Davis, CA, USA, pp 19–24
13. Zhang M, Yang L, He S, Li M, Zhang J (2021) Privacy-preserving data aggregation for mobile crowd-sensing with externality: an auction approach. IEEE ACM Trans Network 29(3):1
14. Krontiris I, Dimitriou T (2013) Privacy-respecting discovery of data providers in crowd-sensing applications. In: IEEE international conference on distributed computing in sensor systems, Cambridge MA, USA, pp 249–257
15. Rohilla A, Khurana M, Singh L (2017) Location privacy using homomorphic encryption over cloud. Int J Comput Sci Net 10(8):32–40
16. Andrés ME, Bordenabe NE, Chatzikokolakis K, et al (2013) Geo-indistinguishability: differential privacy for location-based systems. In: Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security, Xi'an China, pp 901–914
17. Xie Z, Hu L, Huang Y, Pang J (2021) A semiopportunistic task allocation framework for mobile crowdsensing with deep learning. Wirel Commun Mob Comput 2021:6643229:1–6643229:15
18. Kazemi L, Shahabi C (2012) Geocrowd: enabling query answering with spatial crowdsourcing. In: Proceedings of the 20th international conference on advances in geographic information systems, Redondo Beach, CA, USA, USA, pp 189–198
19. Wang L, Yang D, Han X, Wang T, Zhang D, Ma X (2017) Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation. In: Proceedings of the 26th international conference on world wide web, Perth, Australia, pp 627–636
20. Wang Z, Hu J, Zhao J, Yang D, Chen H, Wang Q (2018) Pay on-demand: dynamic incentive and task selection for location-dependent mobile crowdsensing systems. In: IEEE 38th international conference on distributed computing systems (ICDCS), Vienna, Austria, pp 611–621
21. Gong W, Zhang B, Li C (2017) Location-based online task scheduling in mobile crowdsensing. In: GLOBECOM 2017–2017 IEEE Global Communications Conference, Singapore, pp 1–6
22. Khaluf Y, Birattari M, Hamann H (2014) A swarm robotics approach to task allocation under soft deadlines and negligible switching costs. In: International conference on simulation of adaptive behavior. Springer, Cham, pp 270–279
23. Ross SM (2013) Applied probability models with optimization applications. Courier Corporation, New York
24. Vilaplana J, Solsona F, Teixidó I, Mateo J, Abella F, Rius J (2014) A queuing theory model for cloud computing. J Supercomput 69(1):492–507
25. Jiao Z, Tian G (2017) The Blocking Lemma and strategy-proofness in many-to-many matchings. Games Econ Behav 102:44–55

26. Gale D, Shapley LS (1962) College admissions and the stability of marriage. Am Math Mon 69(9):5
27. Roth AE (2008) Deferred acceptance algorithms: history, theory, practice, and open questions. Int J Game Theory 36(3):537–569
28. Wang Z, Hu J, Lv R, Wei J, Wang Q, Yang D, Qi H (2018) Personalized privacy-preserving task allocation for mobile crowdsensing. IEEE Trans Mobile Comput 18(6):1330–1341
29. Gu Y, Saad W, Bennis M, Debbah M, Han Z (2015) Matching theory for future wireless networks: fundamentals and applications. IEEE Commun Mag 53(5):52–59
30. Xu H, Li B (2011) Seen as stable marriages. In: Proceedings IEEE INFOCOM, Shanghai, China, pp 586–590
31. Hamidouche K, Saad W, Debbah M (2014) Many-to-many matching games for proactive social-caching in wireless small cell networks, In 2014 12th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks(WiOpt). IEEE, Hammamet, Tunisia, pp 569–574
32. Zheng T, Chen J, Huang Y (2019) Task assignment in business processes based on completion rate evaluation. In: International conference on computer, network, communication and information systems (CNCI 2019). Atlantis Press, pp 326–333