J. Parallel Distrib. Comput. 71 (2011) 822–836

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Locality-sensitive task allocation and load balancing in networked multiagent systems: Talent versus centrality

Yichuan Jiang [a,b,*], Zhaofeng Li [c]

[a] *Key Laboratory of Computer Network and Information Integration of State Education Ministry, School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*

[b] *State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an 710054, China*

[c] *School of Computer Science and Engineering, Southeast University, Nanjing 211189, China*

ARTICLE INFO

Article history:
Received 10 January 2010
Received in revised form
13 January 2011
Accepted 14 January 2011
Available online 26 January 2011

Keywords:
Multiagent systems
Network structures
Task allocation
Load balancing
Locality-sensitive

ABSTRACT

With the development of large scale multiagent systems, agents are always organized in network structures where each agent interacts only with its immediate neighbors in the network. Coordination among networked agents is a critical issue which mainly includes two aspects: task allocation and load balancing; in traditional approach, the resources of agents are crucial to their abilities to get tasks, which is called *talent-based allocation*. However, in networked multiagent systems, the tasks may spend so much communication costs among agents that are sensitive to the agent localities; thus this paper presents a novel idea for task allocation and load balancing in networked multiagent systems, which takes into account both the talents and centralities of agents. This paper first investigates the comparison between talent-based task allocation and centrality-based one; then, it explores the load balancing of such two approaches in task allocation. The experiment results show that the centrality-based method can reduce the communication costs for single task more effectively than the talent-based one, but the talent-based method can generally obtain better load balancing performance for parallel tasks than the centrality-based one.

1. Introduction

A networked multiagent system can be shaped as a graph consisting of agents (vertices) and interaction relations (edges). Nowadays, with the development of large scale multiagent systems, agents are always organized in network structures where each agent interacts only with its immediate neighbors in the networks [1,8]. For example, in a multiagent system simulating corporation organization, the agents (which denote the people in the corporation) can be organized by a network where each agent only interacts with its immediate superiors, immediate underlings and neighboring colleagues. Therefore, no matter how large the corporation organization is, the agents only need to know the local information in the network, which can significantly reduce the complexity of system design.

In the networked multiagent systems, the tasks are implemented by the collaboration of a large number of agents [19,12]; therefore, coordination among agents is considered as one of the key issues to implement tasks effectively, which mainly includes two aspects: *task allocation* and *load balancing* [10,25,11,22,18,4,13,6,16,23].

When the tasks come to a multiagent system, the first step is to allocate the tasks to certain agents, which is called *task allocation* [19,12,10,25,11,22]. Agent networks consist of numerous agents with different resources [24,17]; in different applications, resources to be placed in agent networks may have various meanings [21]. For example, in a content distribution network, the resources refer to the replicas of popular Web objects; in a multiagent hotel-booking system, the resources refer to the data of hotel information. Generally, the task allocation in previous related work is always implemented based on resources; the number of allocated tasks on an agent is always directly proportional to its *resources* [22,18]. The resources owned by an agent are called its talents, thus *the traditional task allocation is based on agent talents*. In the talent-based task allocation, when a task needs a resource unit, it may in many cases find an available unit in an ideal agent who has such resource; therefore, the agents with more available resources will have higher priority to obtain tasks.

However, in a networked multiagent system, each agent interacts only with its immediate neighbors; thus in some applications it is preferable if agents are able to coordinate with nearby neighbors for implementing tasks, which can reduce the communication costs among agents. Therefore, now it may be better if the tasks are allocated to the agents that are near to each

* Corresponding author at: Key Laboratory of Computer Network and Information Integration of State Education Ministry, School of Computer Science and Engineering, Southeast University, Nanjing 211189, China.

E-mail addresses: jiangyichuan@yahoo.com.cn, yjiang@seu.edu.cn (Y. Jiang), lizhaofeng@live.cn (Z. Li).

0743-7315/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2011.01.006

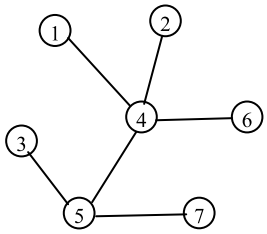


Fig. 1. An example of a networked multiagent system.

other in the network. Now we use an example to illustrate this idea as follows.

Fig. 1 shows a networked multiagent system. Now let a task come to the system whose required resource is r, and the execution of such task be: "*let the set of agents be A, the allocated agent will collect some data from all other agents; after obtaining the data from all other agents, the allocated agent will combine those intermediate results together and use r to processing it*". Now, if only a_7 has r, then the task is allocated to a_7 according to the traditional talent-based method; the total communication costs to execute the task are: $\{\langle a_1\text{–}a_4\text{–}a_5\text{–}a_7\rangle, \langle a_2\text{–}a_4\text{–}a_5\text{–}a_7\rangle, \langle a_3\text{–}a_5\text{–}a_7\rangle, \langle a_4\text{–}a_5\text{–}a_7\rangle, \langle a_5\text{–}a_7\rangle, \langle a_6\text{–}a_4\text{–}a_5\text{–}a_7\rangle\}$. Let the distance between any two neighboring agents be 1, thus the total communication costs are 14 units.

Now, if we allocate the task to a_4, it will first collect the data from other agents so that the total communication costs are $\{\langle a_1\text{–}a_4\rangle, \langle a_2\text{–}a_4\rangle, \langle a_3\text{–}a_5\text{–}a_4\rangle, \langle a_5\text{–}a_4\rangle, \langle a_6\text{–}a_4\rangle, \langle a_7\text{–}a_5\text{–}a_4\rangle\}$. After then a_4 sends the collected data to a_7, and a_7 will use r to process the data; thus this communication cost is $\langle a_4\text{–}a_5\text{–}a_7\rangle$. Therefore, the total communication costs are 10 units if a_4 is allocated with the task. Since high communication costs significantly influence the multiagent system performance [5], it is better to allocate task to a_4 than to a_7 although a_4 does not have the required talent. Thus such task is sensitive to the localities of allocated agents, which is called *locality-sensitive task*.

From the above example, we can see that the localities of agents should be considered in the task allocation of networked multiagent systems; however, most traditional task allocation algorithms do not take locality into consideration. Though [2] presented a locality-sensitive resource allocation model which takes into account the distance between the requesting vertex and the locality of resource, it did not make quantitative comparative analyses between talents and localities of agents. Centrality is one of the most important and widely used conceptual tools for measuring the locality of agent in the network, which denotes how centrally the agent locates in the network [3]. To solve such problem, we investigate the locality-sensitive task allocation in networked multiagents based on comparison *between talent and centrality*, which gains some inspirations from [14] that addresses the balance between experience and talent to shapes the structure of the web.

On the other hand, if too many tasks are crowded on certain agents that are rich of resources, then the tasks may be delayed and do not get a quick response. Therefore, some tasks can be switched to other agents with relatively fewer resources but lower task loads, which is called *load balancing*. In the related work [18, 6,23], if there are too many tasks that queue for an agent, then the probability of such agent to get new tasks will be reduced; the migration of tasks is always implemented according to the resource distribution, i.e., the tasks can be migrated from the agents with rich resources to the agents with relatively poor resources. However, such traditional load balancing method may be not suitable to the locality-sensitive tasks. For example, in Fig. 1, if a_4 is crowded with too many tasks, then it may be better to switch some tasks to a_5 which is near to a_4. Therefore, load balancing should take into account the localities of agents; in the related benchmark work, Chow and Kwok presented a novel communication-based approach to multiagent load balancing, which mainly concerns the *localities of agents in the underlying computer hosts* [4,5]. Now, since the interaction localities of agents are very important in the networked multiagent systems, in this paper we will develop the load balancing method by taking into account the *agent centralities in the multiagent interaction networks*.

The purpose of this paper is to explore the task allocation in networked multiagent system, termed locality-sensitive task allocation and load balancing; the basic idea is to utilize locality, especially centrality, in order to reduce the locality-related costs in the execution of tasks. This paper mainly investigates the comparison between the talent- and centrality-based approaches, and tries to find the effects of such two approaches in different situations. The rest of this paper is organized as follows. In Section 2, we describe the locality-sensitive tasks in networked multiagents; in Section 3, we present the talent-based task allocation and load balancing model; in Section 4, we present the centrality-based task allocation and load balancing model; in Section 5, we present comparative experiments and analyses for the two models; in Section 6, we compare our work with the related work; finally, we conclude our paper in Section 7.

2. Agent cooperation for locality-sensitive tasks

2.1. Cooperation among agents for tasks[1]

In networked multiagent systems, each agent interacts only with its immediate neighbors in the network; communication delay between two agents is proportional to the Euclidean distance between them [1].

When a task comes to a multiagent system, it may be allocated to an agent that will take charge of the implementation of such task (the allocated agent is called principal one). If the principle agent lacks necessary resources to implement the allocated task, we can make it cooperate with other agents; if there are some agents which have the required resources (we call those cooperated agents that provide resources to implement task as *assistant agents*), then the principal agent and assistant agents will cooperate together to implement such task [10]. Obviously, the communication cost between the principal agent and assistant agents is a key for the performance of implementing such task, and which is determined by the localities of and topology among the principal agent and assistant agents. Therefore, now the performance of such task is sensitive to the localities of allocated agents; thus such task is *locality-sensitive*.

Then, which ones will be selected as assistant ones while the principal agent is fixed? To minimize the communication costs, the principal agent can seek the assistant agents gradually from the near to the distance in the network.

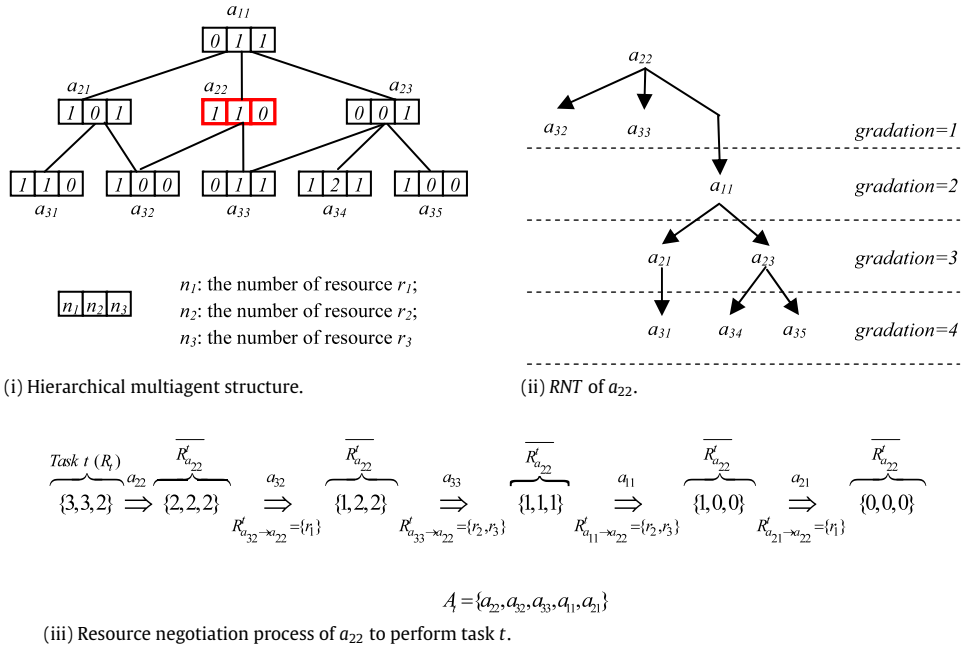
Definition 1 (*Negotiation Gradation*)**.** Let a be the principal agent, the agents in the nth round of negotiation of agent a are called the contexts with gradation n.

Definition 2 (*Resource Negotiation Topology (RNT)*)**.** A *RNT* of agent a is a directed acyclic graph with single source a, the agents in the graph are the ones cooperate with agent a, and the path length from a to any other agents in *RNT* is such agent's negotiation gradation.

Let a_i be the principal agent, and the resources owned by a_i be R_{ai}; now, task t is allocated to a_i, and the set of requested resources for implementing t is R_t. Therefore, the set of lacking resources of a_i to implement t is $\overline{R^t_{a_i}}$:

$$\overline{R^t_{a_i}} = R_t - R_{a_i}. \tag{1}$$

[1] Some contents in Section 2.1 were presented in our previous paper in IEEE TPDS [10].



(i) Hierarchical multiagent structure.

(ii) *RNT* of a_{22}.

(iii) Resource negotiation process of a_{22} to perform task t.

Fig. 2. An example for constructing the resource negotiation topology (*RNT*) in a hierarchical multiagent network structure, in which it is assumed that a_{22} is the principal agent.

Now it is assumed that agent a_j is negotiated by a_i, the set of resources owned by a_j is R_{aj}. If a_j has any resources that are requested by a_i to implement task t, then the set of resources that a_j can provide to a_i for implementing t is:

$$R^t_{a_j \to a_i} = \{r | r \in R_{a_j} \wedge r \in \overline{R^t_{a_i}}\}. \tag{2}$$

Thus now the set of lacking resources of a_i to implement t will be deduced as:

$$\overline{R^t_{a_i}} = \overline{R^t_{a_i}} - R^t_{a_j \to a_i}. \tag{3}$$

We can let the principal agent a_i negotiate with other agents according to the *RNT*, until all requested resources are satisfied.

There are many kinds of network structures of multiagents, among which the hierarchical structure is typical [15]; thus we first address the negotiation of resources in hierarchical structures; after that we will address the one in arbitrary network structures.

(1) Hierarchical structure

In the hierarchical structures, each agent can interact *directly* only to its superiors and subordinates; thus agent will first negotiate with its superiors or subordinates for resources. Moreover, in the hierarchical organizations, resource negotiation is always happened between pairs of actors that share the same immediate superior; and actors will always negotiate resources through the lowest common ancestor [7]. Therefore, let there be an agent, a, we can make it negotiate with other agents according to the following orders:

(a) the subordinates of agent a;
(b) the immediate superior of agent a;
(c) the sibling agents with the lowest common superiors.

Let a be the initiator agent, and the set of agents in the network be A, now the resource negotiation process of agent a in hierarchical structures is shown as Algorithm 1.

We can use the lay-based traversal method in tree to implement the negotiation process in Algorithm 1, shown as Algorithm 2.

Complexity analyses of Algorithm 1 and 2: Let the number of all agents be $|A|$, the height of hierarchical structure be h, now we analyze the complexities of Algorithm 1 and 2, shown as follows.

- On the negotiation time: (1) Algorithm 2 is implemented based on the *lay-based tree traversal method* whose time complexity is $O(|A|)$ [20], thus the complexity of Algorithm 2 is $O(|A|)$; (2) Algorithm 1 is mainly composed by Sentence (6.3) and calling Algorithm 2, thus the complexity of Algorithm 1 is $O(|A| \times h)$.
- On the message overheads: the message overheads are the messages exchanged between the principal agent and all other negotiation agents, which are mainly about the information of resources. Obviously, the message overheads in the network are directly proportional to the communication distance between principal agent and other agents, thus whose complexity is $O(h)$.

Example 1. In Fig. 2, (i) shows a hierarchical multiagent structure, now let a_{22} be the principal agent, we can see the construction of *RNT* according to Algorithm 1, shown as (ii). In Fig. 2, the set of resources owned by agent a_{22} is $\{r_1, r_2\}$. Now a task t is allocated to a_{22}, and the resources requested by task t is $\{3r_1, 3r_2, 2r_3\}$; obviously, a_{22} lacks the resources $\{2r_1, 2r_2, 2r_3\}$. Now a_{22} will make resource negotiation with other agents; the negotiation process is seen in (iii). Finally, the set of agents $\{a_{22}, a_{32}, a_{33}, a_{11}, a_{21}\}$ will cooperate to implement task t.

Obviously, with Algorithm 1 and 2, we can constrain the resource negotiation process in hierarchical structures as follows.

Lemma 1. *If all resources for implementing task t can be satisfied by using Algorithm 1 and 2, the set of assistant agents can satisfy one of the following situations: (1) all assistant agents are located in the subtree of principal agent; or (2) the immediate common ancestor between the principal agent and all assistant agents is the lowest in the tree (i.e., let such ancestor be $a_\#$ and the distance from $a_\#$ to the root of tree be $d_\#$, it is impossible that there exists an agent whose subordinates can provide all required resources for the principal agent and its distance to the tree root is less than $d_\#$).*

According to the information exchange criterion in [7], our presented resource negotiation method has higher probability to reduce the communication cost compared to other random negotiation processes.

(2) Arbitrary structure

To minimize the negotiation communication time, we can make principal agent negotiate with other agents for the requested resources based on the Breadth-First Traversal method in graph. The algorithm for the negotiation process within arbitrary structure is shown as Algorithm 3.

Algorithm 1. Resource negotiation of agent a in a hierarchical structure.
/* T_x: the subtree whose root is agent x in the hierarchical structure; p_x: the parent node of x in the hierarchical structure. */

(1) **Set** the tags for all agents in A to 0 initially;
(2) $b = 0$;
(3) $A_t = \{a\}$; /*The allocated agent set for task t */
(4) $\overline{R_a^t} = R_t - R_a$; /*The lacking resources of agent a to implement task t */
(5) **If** $\overline{R_a^t} == \{\}$ **then** $b = 1$; /*Agent a can provide all resources to implement task t */
(6) **If** $(b == 0)$ **then**:
  (6.1) $b =$ **Negotiation**(a, a); /* a negotiates with the agents within T_a by calling Algorithm 2 */
  (6.2) $atemp = a$;
  (6.3) **While** $((b == 0)$ and $(p_{atemp} <> \text{Nil}))$ **do**:
    (6.3.1) $atemp = p_{atemp}$;
    (6.3.2) $b =$ **Negotiation** $(a, atemp)$; /* a negotiates with the agents within T_{atemp} by calling Algorithm 2 */
(7) **If** $(b == 1)$ **then Return**(A_t) /*All resources for implementing t are satisfied */
  else Return (False);
(8) **End**.

Algorithm 2 Negotiation $(a\ x)$. /* Agent a negotiates with the agents in subtree T_x */

(1) **Create** Queue (Q);
(2) **Insert** Queue (Q, x);
(3) **Set** the tag of x to 1;
(4) **While**(!EmptyQueue(Q) and $(b == 0)$) **do**:
  (4.1) $aout =$ **Out** Queue(Q);
  (4.2) $R' = \overline{R_a^t} - R_{aout}$;
  (4.3) **If** $R' \neq \overline{R_a^t}$ **then**:
    (4.3.1) $\overline{R_a^t} = \overline{R_a^t} - R_{aout}$; /*Agent a obtains resources from $aout$ to implement t */
    (4.3.2) $A_t = A_t \cup \{aout\}$;
  (4.4) **If** $\overline{R_a^t} == \{\}$ **then** $b = 1$; /*All resources for implementing t are satisfied */
  (4.5) **For** $\forall achild \in child(aout)$:
    If the tag of $achild$ is 0:
      (4.5.1) **Insert** Queue(Q,$achild$);
      (4.5.2) **Set** the tag of $achild$ to 1;
(5) **Return** (b);
(6) **End**.

Complexity analyses of Algorithm 3: Let the number of all agents be $|A|$ and the diameter of network be d, now we make analyses on the complexity of Algorithm 3, shown as follows.

On the negotiation time: Algorithm 3 is implemented based on the *Breadth-First Traversal method in graph*, the main parts of Algorithm 3 are Sentence (9) and (9.5), thus the complexity is $O(|A| \times |A|)$.

On the message overheads: the message overheads are the messages exchanged between the principal agent and all other negotiation agents, which are mainly about the information of resources. The message overheads in the network are directly proportional to the communication distance between principal agent and other agents, thus whose complexity is $O(d)$.

Theorem 1. *If all resources for implementing task t can be satisfied by using our algorithm, the total communication costs between the principal agent and assistant agents in arbitrary structure can be minimized.*

Proof. Let a_i be the principal agent, the set of lacking resources of a_i to implement t is $\overline{R_{a_i}^t}$. If Algorithm 3 is used, the set of assistant agents is A_* ($A_* = A_t - \{a_i\}$), and the total communication cost between a_i and A_* is C_*. Now, if there is a set of agents A'_*, $A'_* \neq A_*$, which can provide $\overline{R_{a_i}^t}$, and the total communication cost between a_i and A'_* is C'_*; if $C'_* < C_*$, it denotes that there are any agents with higher gradations that provide the required resources in $\overline{R_{a_i}^t}$, but the lower gradation agents with required resources do not provide the required resources in $\overline{R_{a_i}^t}$. Obviously, such situation cannot take place in Algorithm 3. Therefore, we have Theorem 1. □

Example 2. Fig. 3 is an example to demonstrate the *RNT* in arbitrary multiagent network structure, where it shows the negotiation process of a_4 to implement task t. For the reason of saving space, we do not express the negotiation process in detail.

2.2. Locality measures for tasks

For a task, after it is allocated to a set of agents, the set of agents will occupy certain localities in the network. Thus the locality of task can be measured by the localities of the allocated agents. In this paper, we present three methods to measure the locality of the set of agents for a task.

Definition 3. *Occupancy percentage of allocated agents for task t*, which denotes how much occupancy that the allocated agents of t cover in the networked multiagent system.

$$\Omega_t = |A_t|/|A| \tag{4}$$

where $|A_t|$ denotes the number of allocated agents for t, $|A|$ denotes the number of all agents.

Definition 4. *Mean geodesic distance* between vertex pairs in the allocated agents of task t.

$$l_t = \frac{2}{|A_t|(|A_t|+1)} \sum_{(a_i, a_j \in A_t) \wedge (i \leq j)} d_{ij} \tag{5}$$

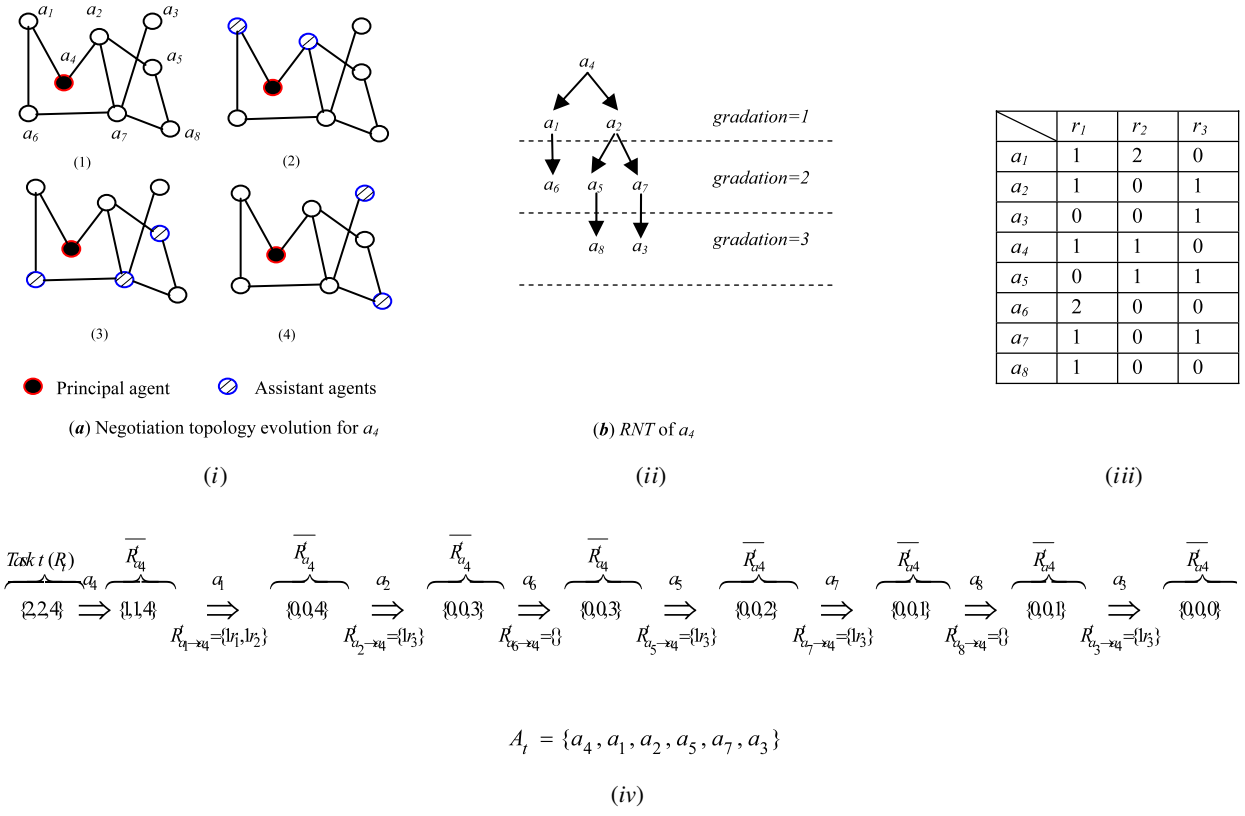
where d_{ij} is the geodesic distance from a_i to a_j, the distance from each agent to itself is also included in this average (which is zero), A_t denotes the allocated agents for t and $|A_t|$ denotes the number of A_t.

Definition 5. *Geodesic center of the networked multiagents*. For a given multiagent network $G = \langle A, E \rangle$, the communication sum cost for every agent $a_i \in A$ is:

$$\mathbb{C}_i = \sum_{a_j \in A} d_{ij}. \tag{6}$$

Algorithm 3. Resource negotiation of agent a in arbitrary structure:
/*a is the principal agent, A is the set of all agents */

(1) **Set** the tags for all agents in A to 0 initially;
(2) **Create** Queue(Q);
(3) **Insert** Queue (Q, a);
(4) **Set** the tag of a to 1;
(5) $b = 0$;
(6) $A_t = \{a\}$; /*The allocated agent set for task t */
(7) $\overline{R_a^t} = R_t - R_a$; /*The lacking resources of agent a to implement task t */
(8) **If** $\overline{R_a^t} == \{\}$ **then** $b = 1$; /*Agent a can provide all required resources to implement task t */
(9) **While** ((!EmptyQueue (Q)) and ($b == 0$)) **do**:
  (9.1) $aout =$ **Out**Queue(Q);
  (9.2) $R' = \overline{R_a^t} - R_{aout}$;
  (9.3) **If** $R' \neq \overline{R_a^t}$ **then**: /*Agent $aout$ can satisfy some requests of a */
    (9.3.1) $\overline{R_a^t} = \overline{R_a^t} - R_{aout}$; /* Agent a obtains resources from $aout$ to implement t */
    (9.3.2) $A_t = A_t \cup \{aout\}$;
  (9.4) **If** $\overline{R_a^t} == \{\}$ **then** $b = 1$; /*All resources for implementing t are satisfied */
  (9.5) **For** $\forall alocal \in L_{aout}$: /* L_{aout} is the set of neighbors of $aout$ */
    if the tag of $alocal$ is 0, **then**: /*If agent $alocal$ was not negotiated by a before */
      (9.5.1). **Insert** Queue (Q, $alocal$);
      (9.5.2). **Set** the tag of $alocal$ to 1;
(10) **If** ($b == 1$) **then Return** (A_t) /* All resources for implementing t are satisfied */
  else Return (False);
(11) **End**.



	r_1	r_2	r_3
a_1	1	2	0
a_2	1	0	1
a_3	0	0	1
a_4	1	1	0
a_5	0	1	1
a_6	2	0	0
a_7	1	0	1
a_8	1	0	0

Fig. 3. An example for constructing the resource negotiation topology (*RNT*) in arbitrary structure: (i) An arbitrary network structure of multiagents and the negotiation topology evolution; (ii) Resource negotiation topology; (iii) Number of resources owned by agents; (iv) Resource negotiation process.

Then the geodesic center of networked multiagents is defined as:

$$a_c = \arg\min_{a_i \in A}(\mathbb{C}_i). \tag{7}$$

Definition 6. *Mean geodesic eccentricity* of the allocated agents of task t can be defined as:

$$\omega_t = \left(\sum_{a_i \in A_t} d_{ic}\right) / |A_t|. \tag{8}$$

Example 3. In Fig. 4, $A_t = \{a_4, a_1, a_2, a_5, a_7\}$, it is assumed that the distance between any two neighboring agents is 1. According to Definitions 3 and 4, $\Omega_t = 5/8$, $l_t = 1.13$. From Fig. 4, we can see that a_7 is the center, thus the mean geodesic eccentricity $\omega_t = 1.4$.

3. Task allocation and load balancing based on talents

3.1. Talent

The talent of an agent is measured by the likelihood that the agent can obtain the resources for implementing tasks.

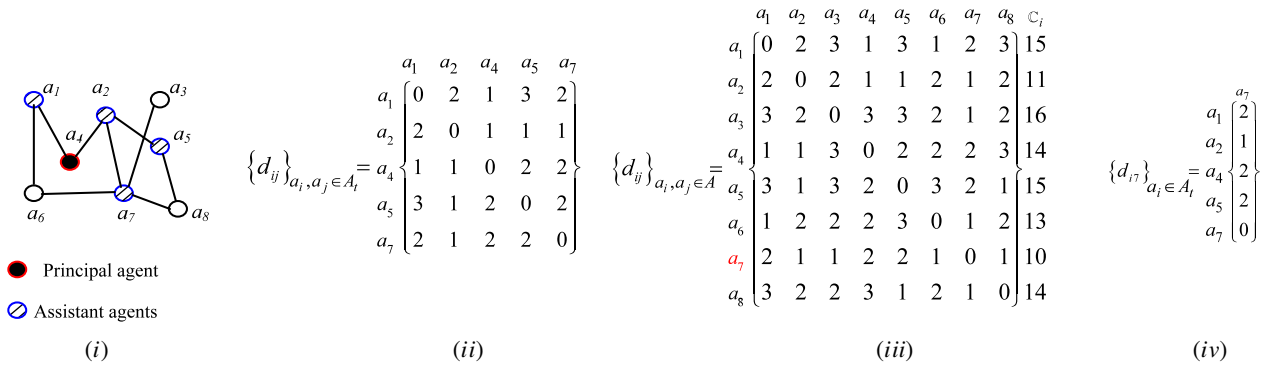


Fig. 4. An example for the locality measure of task: (i) The network structure of multiagents and the allocated agents of task t; (ii) The matrix of distances between vertex pairs in the allocated agents of task t; (iii) The matrix of distances between vertex pairs of all agents; (iv) The matrix of distance between each allocated agent and the center in the network.

Table 1
The talents of agents in Fig. 3 ($\lambda_{in} = \lambda_{ex} = 0.5$).

Resources	Talents	Agents							
		a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
r_1	$\phi_i(1)$	1/7	1/7	0	1/7	0	2/7	1/7	1/7
	$\theta_i(1)$	29/84	1/3	1/3	9/28	29/84	2/7	17/42	2/7
	$\omega_i(1)$	41/162	5/21	1/6	13/56	29/168	2/7	23/84	3/14
r_2	$\phi_i(2)$	1/2	0	0	1/4	1/4	0	0	0
	$\theta_i(2)$	3/16	1/2	1/4	1/3	5/24	19/48	1/3	5/16
	$\omega_i(3)$	11/32	1/4	1/8	7/24	11/48	19/96	1/6	5/32
r_3	$\phi_i(3)$	0	1/4	1/4	0	1/4	0	1/4	0
	$\theta_i(2)$	7/24	1/3	13/48	17/48	13/48	17/48	1/3	5/12
	$\omega_i(3)$	7/48	7/24	25/96	17/96	25/96	17/96	7/24	5/24

Definition 7 (*Inherent Talent of Agent*)**.** The talent of an agent is defined by comparing its resources with other agents' resources. Let there be a resource r_k, the number of r_k in the whole system be $n(k)$, and the number of r_k owned by a_i be $n_i(k)$, then the inherent talent on resource r_k of agent a_i is:

$$\phi_i(k) = n_i(k)/n(k). \tag{9}$$

In the networked multiagent system, an agent can negotiate and *borrow* resources from other agents according to Section 2.1. Thus an agent's talent can also be influenced by its contextual resources [10]. Now we have the following definition.

Definition 8 (*Extrinsic Talent of Agent*)**.** Let $g_i(j)$ denote the negotiation gradation of agent a_j in the *RNT* of a_i, Z_i denotes the set of agents in the *RNT* of a_i. The extrinsic talent of a_i for r_k is:

$$\theta_i(k) = \sum_{j \in Z_i} \phi_j(k)/(g_i(j) + 1). \tag{10}$$

The denominator in (10) is $g_i(j) + 1$, which can differentiate the influences of the agent itself and its immediate neighbors.

Definition 9. *Comprehensive talent of agent* can be defined as:

$$\omega_i(k) = \lambda_{in}\phi_i(k) + \lambda_{ex}\theta_i(k) \tag{11}$$

where λ_{in} and λ_{ex} are to determine the relative importance of the two kinds of factors in the comprehensive talent of an agent, $\lambda_{in} + \lambda_{ex} = 1$.

Example 4. Now we can take the multiagent system in Fig. 3 as an example to compute the talents of agents, the results are shown in Table 1. From Table 1, we can see that $\phi_1(2) > \phi_2(2)$, but $\theta_1(2) < \theta_2(2)$; it denotes that agent a_2 can easily obtain r_2 from other agents than a_1, though a_2 has less r_2s than s_1.

3.2. Task allocation

3.2.1. Case for single resource

In some cases, a task may only need single kind of resources. For example, in Fig. 3 and Table 1, let there be a task that needs resource r_1. If we implement task allocation based on the inherent talents of agents, we should allocate the task to agent a_6; if we implement task allocation based on the extrinsic talents of agents, we should allocate the task to agent a_7; if we implement task allocation based on the comprehensive talents of agents, we should allocate the task to agent a_6.

3.2.2. Case for multiple resources

When a task requires many resources, it may call each resource sequentially. Now we can design the criterion of *FRFS* (*First required resource-first satisfy*), i.e., the agent that has the highest talent for the first called resource should be allocated as the principal one of that task. Let there be a task t, the set of resources called by task t orderly is $\{n_1r_1, n_2r_2, \ldots, n_nr_n\}$, n_i denotes the required number of resource r_i. So we will allocate t to agent a_i which has the highest talent for r_1, i.e., the highest $\phi_i(1)$, $\theta_i(1)$, or $\omega_i(1)$. For example, in Fig. 3 and Table 1, if the called resource sequence of a task is $\{r_2, r_1, r_2, r_3\}$, and the task allocation is implemented based on comprehensive talents of agents, $a_1$will be assigned as the principal one for such task.

3.2.3. Task allocation process

Now we have two methods for task allocation, one is semi-supervised allocation, the other one is full-supervised allocation.

In the semi-supervised allocation, we should assign an agent to act as the *principal one* which will initially negotiate with other agents for the resources required by the task; then the set of all agents that provide resources for the task is the allocated

Algorithm 4. Semi-supervised allocation based on agent talents.

(1) **Select** the principal agent, a_*, according to the resource satisfaction criteria in Section 3.2.1 or 3.2.2;
(2) **Case** "*Network structure*" of
"*Hierarchical structure*": a_* negotiates with other agents for required resources by calling Algorithm 1;
"*Arbitrary structure*": a_* negotiates with other agents for required resources by calling Algorithm 3;
End;
(3)**Output** A_t;
(4) **End**.

agents. The algorithm of semi-supervised allocation is shown as Algorithm 4.

In the full-supervised allocation, each resource is satisfied according to the talents. Let a_i be an agent, and the resources owned by a_i be R_{ai}; now, task t comes to the system, and the set of requested resources for implementing t is R_t, the set of lacking resources to implement t is $\overline{R_t}$. The algorithm of full-supervised allocation is shown as Algorithm 5.

From the two algorithms, we can see that full-supervised allocation algorithm looks for the agent with the highest talent for each required resource, but the semi-supervised allocation algorithm only looks for the principal agent with the highest talent and the principal agent will seek for the other assistant agents.

Theorem 2. *If the full-supervised allocation algorithm based on agent talents is used, the number of agents allocated on any tasks can be minimized.*

Proof. Let task t come to the system; if the full-supervised allocation based on agent talents is used, the set of allocated agents is A_t. Now, if there is a set of agents A'_t, $A'_t \neq A_t$, which can provide R_t; if $|A'_t| < |A_t|$, it denotes that there are any agents with higher talents for resource in R_t that are not allocated with such task, but the agents with lower talents for resource in R_t are allocated with such task; obviously, such situation cannot take place in the full-supervised allocation based on agent talent where all allocated agents are selected according to their talents. Therefore, we have Theorem 2. □

Therefore, the number of allocated agents will be reduced by the full-supervised allocation since it can select the highest talent agents for each required resource, thus the resources can be satisfied by agents as few as possible, accordingly the reduced number can also reduce the communication costs; on the other hand, the communication costs can be reduced by the semi-supervised allocation from Lemma 1 and Theorem 1. *Therefore, these two algorithms can both reduce the communication costs than other random allocation methods in practice.*

3.3. Load balancing

3.3.1. The model

An agent may have more tasks to queue if it has higher talents. Therefore, we should deal with the *load balancing* while there are multiple tasks within the multiagent system.

Generally, the performance of load balancing is mainly determined by the waiting time of task. According to [18], the effect of measures on load balancing is reflected by the number and size of task teams on agents; and we simply consider that the waiting time is only related to the length of task team. Let the allocated agent set of task t be A_t, the set of resources required by t be R_t. We can denote the team of tasks that queue for resource r_k of agent a_i as Q_{ik}, and the size of Q_{ik} is $|Q_{ik}|$. If task t calls its requested resources concurrently, the waiting time of task t will be determined by the maximum size of the queues for its required resources; if task t calls its requested resources serially, the waiting time of task t will be determined by the total sizes of the queues for its required resources. Therefore, we can define the waiting time of a task as:

$$w_t = \max_{\forall r_i \in R_t, \forall a_k \in A_t} (|Q_{ik}|), \quad \text{or} \quad w_t = \sum_{\forall r_i \in R_t, \forall a_k \in A_t} (|Q_{ik}|). \tag{12}$$

Now, if there are multiple tasks come to a multiagent system, the set of those multiple tasks is T, we can define the global load balancing performance of the system is:

$$\widetilde{w} = \frac{1}{|T|} \sum_{t \in T} w_t. \tag{13}$$

Therefore, our goal of load balancing is to reduce the average size of the queues for all resources and all agents. In previous research work on load balancing [18,13], an agent's probability to be allocated new tasks will be reduced if it has already undertaken too many tasks; thus the number of undertaken tasks of an agent will only influence the probability of itself to be allocated new tasks in the future.

In the networked multiagent systems, the agents may cooperate to implement tasks according to the network structure [9]. Therefore, the number of undertaken tasks of an agent will also influence the probability of its cooperated agents to be allocated new tasks in some degrees.

Definition 10. *The infection sub-structure of an agent in the network* is defined as the set of interaction relations linking this agent with other agents in the network. Let the network structure of multiagents be $G = \langle A, E \rangle$, where $A = \{a_1, a_2, \ldots, a_n\}$ denotes the set of agents and E denotes the set of agent networking relations. Then the *1st-order* infection sub-structure of an agent, $a_i \in A$, is the union of its immediate networking links:

$$Inf_{a_i} = \{(a_j, a_i) | a_j \in A \wedge (a_j, a_i) \in E\}. \tag{14}$$

Obviously, the *2nd-order* infection sub-structure of a_i can be defined as:

$$\begin{aligned} \mathrm{Inf}(\mathrm{Inf}_{a_i}) = \{&(a_k, a_j) | a_j \in A \wedge a_k \in A \wedge (a_j, a_i) \\ &\in E \wedge (a_k, a_j) \in E\}. \end{aligned} \tag{15}$$

Therefore, the *nth-order* infection sub-structure of a_i can be defined as:

$$\begin{aligned} \prod_n Inf_{a_i} &= \overbrace{Inf\,(Inf\,(\ldots\,(Inf_{a_i})\ldots))}^{n} \\ &= \{(a_n, a_{n-1}) | a_1 \in A \wedge a_2 \in A \wedge \ldots \wedge a_n \in A \wedge (a_n, a_{n-1}) \\ &\quad \in E \wedge \ldots \wedge (a_2, a_1) \in E \wedge (a_1, a_i) \in E\}. \end{aligned} \tag{16}$$

The set of agents within the *1st-order* infection sub-structure of agent a_i (called its *1st-order infection agents*) is:

$$\measuredangle_{a_i} = \{a_j | a_j \in A \wedge (a_j, a_i) \in \mathrm{Inf}_{a_i}\}. \tag{17}$$

Let $a_j \odot r$ denote that the networking relation r is associated with agent a_j, the set of all agents within the *all-orders* infection sub-structures of agent a_i is:

$$\sum \measuredangle_{a_i} = \bigcup_k \left\{ a_j | a_j \odot r \wedge r \in \prod_k \mathrm{Inf}_{a_i} \right\}. \tag{18}$$

Example 5. Fig. 5 is an example for the infection sub-structures.

Algorithm 5. Full-supervised allocation based on agent talents.
/* A is the set of all agents, A_t is the set of agents to implement task t */

(1) $\overline{R_t} = R_t$;
(2) $A_t = \{\}$;
(3) **While** $\overline{R_t} \neq \{\}$ do:
  (4.1) For $\overline{R_t}$: **Select** the principal agent, a_*, according to the resource satisfaction criteria in Section 3.2.1 or 3.2.2;
  (4.2) $A_t = A_t \cup \{a_*\}$;
  (4.3) $\overline{R_t} = \overline{R_t} - R_{a_*}$;
(5) **If** $(\overline{R_t} == \{\})$ **then Return**(A_t) /* All resources for implementing t are satisfied */
  else Return (False); ;
(6) **End**.

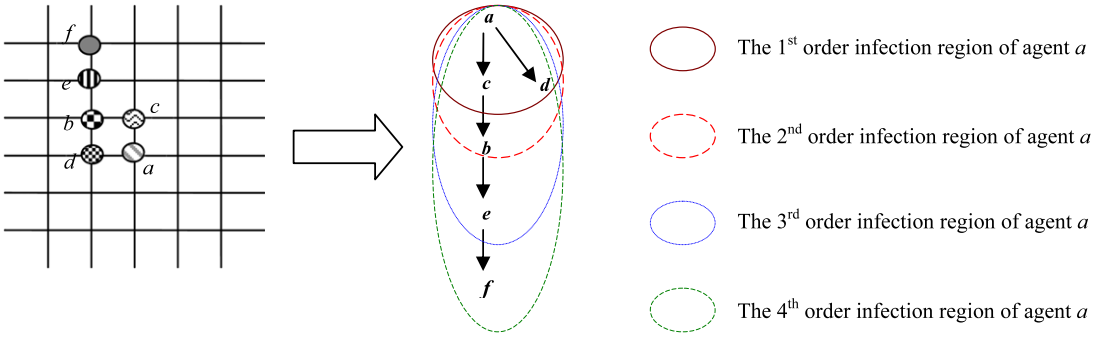


Fig. 5. An example of the infection sub-structure: (i) a multiagent network structure; (ii) the infection sub-structure of agent a.

Since an agent will cooperate with other agents in the network structure for resources; thus if an agent has already been allocated with too many tasks, the probabilities of its infected agents to accept new tasks should also be reduced. Thus we will develop the load balancing method also based on the infection sub-structure. If an agent, a, is allocated with some tasks, it is not only a itself but also the agents in the infection sub-structure of a will be reduced the possibilities to accept new tasks in the future. Let $|Q_{ik}|$ be the size of team where tasks queue for resource r_k of agent a_i, the extrinsic talents of all agents in the all-orders infection sub-structures of a_i should be changed as:

$$\theta_i(k) = \theta_i(k) - \sigma_1(Q_{ik})$$
$$\forall a_j \in \sum \measuredangle_{a_i}, \quad \theta_j(k) = \theta_j(k) - \sigma_2(Q_{ik}/d_{ij}) \tag{19}$$

where σ_1, σ_2 are two monotonously increasing functions, d_{ij} denotes the order of a_j in the infection sub-structure of a_i.

For the inherent talent, only the one of a_i itself should be changed, shown as:

$$\phi_i(k) = \phi_i(k) - \sigma_3(Q_{ik}) \tag{20}$$

where σ_3 is a monotonously increasing function.

3.3.2. A case study

Now we make a case study on the load balancing based on inherent talent, i.e. the load balancing is implemented by Eq. (20); for the reason of demonstrating the case clearly, we can let $\sigma_3(Q_{ik}) = Q_{ik}$, $\phi_i(r) = n_i(k)$, and the all tasks share the same execution time. Now let there be n tasks, the execution time of each task is η, the resource required for each task is $1r$ (i.e., the number of resource r is 1); if task t is allocated to agent a_i, thus the real finish time of t is $Q_{ir}\eta$. Now there are m agents, $a_1, a_2, \ldots, a_m$; The inherent talents of agents descending from a_1 to a_m monotonously, i.e, a_1 has the maximum inherent talent, a_m has the minimum inherent talent; the difference of inherent talents between a_i and a_{i+1} is $\gamma_{i,i+1}$. We make task allocation on the inherent talents of agents.

(1) If we make task allocation without considering load balancing, the n tasks will be all allocated to a_1, thus the finish time of task t_i is $i\eta$. Thus the global performance of system is

$$\widetilde{w} = \left(\sum_{t_i \in T} i\eta\right) \Big/ |T| \tag{21}$$

where T is the set of all tasks, and $|T|$ is the number of all tasks.

(2) Now we make task allocation considering load balancing, thus we have the following allocation results:

(2.1) From the 1st to the $(1 + \gamma_{1,2})$th task: all $\gamma_{1,2}$ tasks are allocated to a_1;

(2.2) From the $(1+\gamma_{1,2}+1)$th task to the $(1+\gamma_{1,2}+1+\gamma_{2,3})$th task: the number of tasks is $\gamma_{2,3}$, among which $\gamma_{2,3}/2$ tasks is allocated to a_1, $\gamma_{2,3}/2$ tasks is allocated to a_2.

(2.3) From the $(1+\gamma_{1,2}+1+\gamma_{2,3}+1)$th task to the $(1+\gamma_{1,2}+1+\gamma_{2,3}+1+\gamma_{3,4})$th task: the number of tasks is $\gamma_{3,4}$, among which $\gamma_{3,4}/3$ tasks is allocated to a_1, $\gamma_{3,4}/3$ tasks is allocated to a_2, $\gamma_{3,4}/3$ tasks is allocated to a_3.

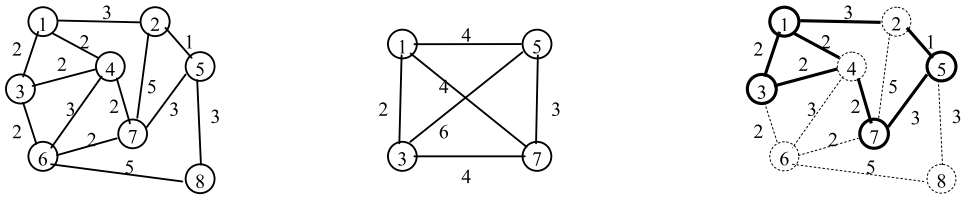
(2.4) For $i = 1$ to $|T|$, from the $(\sum_{x=1,\ldots,i} \gamma_{x,x+1} + i + 1)$th to the $(\sum_{x=1,\ldots,s,i+1} \gamma_{x,x+1} + i + 1)$th task: the number of tasks is $\gamma_{i+1,i+2}$, then those $\gamma_{i+1,i+2}$ tasks should be divided evenly into $i + 1$ parts and each part of tasks are allocated to a_j $(1 \leq j \leq i + 1)$.

(2.5) Finally, the number of tasks allocated to agent a_i is: $|T_i| = \sum_{x=i,\ldots,|T|} \gamma_{x,x+1}/x$.

(2.6) For each agent a_i, the average waiting time of the tasks allocated on a_i is: $\widetilde{w}_i = (\sum_{y=1,\ldots,|T_i|} y\eta)/|T_i|$.

(2.7) Therefore, the global performance of the system is:

$$\widetilde{w} = \left(\sum_{i=1,\ldots,m} \widetilde{w}_i\right) \Big/ m = \left(\sum_{i=1,\ldots,m} \left(\left(\sum_{y=1,\ldots,|T_i|} y\eta\right) \Big/ |T_i|\right)\right) \Big/ m$$
$$= \left(\sum_{i=1,\ldots,m} \left(\left(\sum_{y=1,\ldots,\left(\sum_{x=i,\ldots,|T|} \gamma_{x,x+1}/x\right)} y\eta\right) \Big/ \left(\sum_{x=i,\ldots,|T|} \gamma_{x,x+1} \Big/ x\right)\right)\right) \Big/ m. \tag{22}$$



(i) (ii) (iii)

Agents	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
C_a	1/26	1/29	1/28	1/27	1/27	1/28	1/25	1/40
C_a^t	1/10	/	1/12	/	1/13	/	1/11	/

(iv)

Fig. 6. An example of *RON*: (i) the network structure of multiagents; (ii) the lengths of shortest paths among agents in R_t; (iii) the final *RON*of t; (iv) the two kinds of centralities.

From Eqs. (21) and (22), we can see that the load balancing can obtain more advantage when the number of tasks increases. For saving space, we do not give the numerical simulation for the two equations.

4. Task allocation and load balancing based on centralities

4.1. Centrality

4.1.1. Centrality in the network structure of agents

Centrality is one of the most important and widely used conceptual tools for analyzing network structures [3]. We can use the closeness to measure the centrality of agent.

Definition 11 (*Centrality of Agent*)**.** An agent is considered important if it is relatively close to all other agents. *Closeness* is based on the inverse of the distance of each agent to every other agent in the network:

$$C_c(a_i) = \left[\sum_{j=1}^{|A|} d(a_i, a_j)\right]^{-1}. \tag{23}$$

Therefore, the probability of an agent that is allocated with task can be defined as the monotonously increasing function of such agent's centrality, shown as follows:

$$\mu(a_i) = \sigma_4(C_c(a_i)) \tag{24}$$

where σ_4 is a monotonously increasing function.

4.1.2. Centrality in the resource overlay sub-network of task

Definition 12 (*Resource Overlay Sub-network of Task (RON)*)**.** Let there be a multiagent network structure $G = \langle A, E\rangle$; for $\forall a \in A$, the set of resources owned by a is R_a; now a task t comes to the network, and the set of resources required by t is R_t. The resource overlay sub-network of the resources of task t, G_{Rt}, is composed of the agents that have any resources in R_t and the shortest paths among those agents. $G_{Rt} = \langle A_{Rt}, E_{Rt}\rangle$, $A_{Rt} = \{a_i | (a_i \in A) \wedge (R_{ai} \cap R_t \neq \emptyset)\}$, $E_{Rt} = \{P_{ij} | a_i, a_j \in A_{Rt}\}$, where P_{ij} denotes the shortest path between a_i and a_j in G.

Fig. 6(i) shows a multiagent network. Now let a task t come to the system, and the resources required by t is R_t, the set of agents that can provide any resources in R_t is $A_{Rt} = \{a_1, a_3, a_5, a_7\}$. We first compute the shortest path between every two agents in A_{Rt}; they are $P_{13} = \{\langle 1, 3\rangle\}$, $P_{15} = \{\langle 1, 2\rangle, \langle 2, 5\rangle\}$, $P_{17} = \{\langle 1, 4\rangle, \langle 4, 7\rangle\}$, $P_{35} = \{\langle 3, 1\rangle, \langle 1, 2\rangle, \langle 2, 5\rangle\}$, $P_{37} = \{\langle 3, 4\rangle, \langle 4, 7\rangle\}$, $P_{57} = \{\langle 5, 7\rangle\}$. Finally, the *RON* of task t is shown as the overstriking part in Fig. 6(iii).

Therefore, we can also compute the centrality of the agents in A_{RT} in *RON* of t, G_t; the centrality in G_t is called the centrality in *RON*.

4.2. Task allocation

Now we also develop two methods for task allocation based on agent centralities, one is semi-supervised allocation, the other one is full-supervised allocation.

In the semi-supervised allocation based on agent centralities, we should assign an agent to act as the *principal one* which will initially negotiate with other agents for the resources required by task; then the set of all agents that provide resources for the task is the allocated agents. On the principal agent, we can select the one with the maximum centrality in G or G_t:

$$a_* = \underset{a \in A_{Rt}}{\operatorname{argmax}}(C_a), \quad \text{or} \quad a_* = \underset{a \in A_{Rt}}{\operatorname{argmax}}(C_a^t). \tag{25}$$

The algorithm of semi-supervised allocation based on centralities is shown as Algorithm 6.

In the full-supervised allocation, each resource is satisfied according to the centralities. Let a_i be an agent, and the resources owned by a_i be R_{ai}; now, task t comes to the system, and the set of requested resources for implementing t is R_t, the set of lacking resources to implement t is $\overline{R_t}$.

Theorem 3. *If the full-supervised allocation method based on agent centralities is used, the average centrality of agents allocated on any tasks can be maximized.*

Proof. Let task t come to the system; if the full-supervised allocation based on agent centralities is used, the set of allocated agents is A_t. Now, if there is a set of agents A'_t, $A'_t \neq A_t$, which can provide R_t; if the average centrality in A'_t is higher than the one of A_t, it denotes that there are any agents with higher centralities that are not allocated with such task, but the agents with lower centralities are allocated with such task; obviously, such situation cannot take place in the full-supervised allocation based on agent centralities where all allocated agents are selected according to their centralities. Therefore, we have Theorem 3. □

Therefore, the centralities of allocated agents will be enhanced by the full-supervised allocation since it can select the highest centrality agents each time, thus it is highly probable that the communication costs can be reduced; on the other hand, the communication costs can be reduced by the semi-supervised allocation from Lemma 1 and Theorem 1. Therefore, *these two algorithms can both reduce the communication costs more probably than other random allocation methods in practice.*

Algorithm 6. Semi-supervised allocation based on agent centralities.

(1) **Construct** the *resource overlay sub-network* (RON) of task t, G_{Rt}, according to Section 4.1.2;
(2) **For** $\forall a \in A_{Rt}$: compute the two kinds of centralities of a, C_a and C_a^t;
(3) **Select** the principal agent, a_*, from A_{Rt}, according to Eq. (25);
(4) **Case** "*Network structure*" of
"*Hierarchical structure*": a_* negotiates with other agents for required resources by calling Algorithm 1;
"*Arbitrary structure*": a_*negotiates with other agents for required resources by calling Algorithm 3;
End;
(5) **Output** A_t;
(6) **End**.

Algorithm 7. Full-supervised allocation based on agent centralities.
/* A is the set of all agents, A_t is the set of agents to implement task t */

(1) $\overline{R_t} = R_t$;
(2) $A_t = \{\}$;
(3) **While** $\overline{R_t} \neq \{\}$ **do**:
(4.1) **Construct** the *resource overlay sub-network* (RON) of $\overline{R_t}$, $G_{\overline{R_t}}$, according to Section 4.1.2;
(4.2) **Select** the principal agent, a_*, from $A_{\overline{R_t}}$, according to Eq. (25);
(4.2) $A_t = A_t \cup \{a_*\}$;
(4.3) $\overline{R_t} = \overline{R_t} - R_{a_*}$;
(5) **If** $(\overline{R_t} == \{\})$ **then Return**(A_t) /* All resources for implementing t are satisfied */
else Return (False);
(6) **End**.

4.3. Load balancing

4.3.1. Semi-supervised allocation

While we use the semi-supervised allocation based on agent centralities, we can develop the load balancing method also based on the infection sub-structure which is similar to the one of Section 3.3. If an agent, a, is allocated with some tasks, it is not only a itself but also the agents in the infection sub-structure of a will be reduced the possibilities to accept new tasks in the future. Let $|Q_{ik}|$ be the size of team where tasks queue for the resource r_k of agent a_i, the agents in the all-orders infection sub-structures of a_i should be changed as:

$$\begin{aligned} &\mu_i(k) = \mu_i(k) - \sigma_5(Q_{ik}) \\ &\forall a_j \in \sum \lhd_{a_i}, \quad \mu_j(k) = \mu_j(k) - \sigma_6(Q_{ik}/d_{ij}) \end{aligned} \tag{26}$$

where σ_5, σ_6 are two monotonously increasing functions, d_{ij} denotes the order of a_j in the infection sub-structure of a_i.

4.3.2. Full-supervised allocation

Let the full-supervised allocation be used; if an agent, a, is allocated with some tasks, it is only a itself that will be reduced the probability to accept new tasks in the future. Let $|Q_{ik}|$ be the size of team where tasks queue for the resource r_k of agent a_i, the probability of a_i should be changed as:

$$\mu_i(k) = \mu_i(k) - \sigma_5(Q_{ik}). \tag{27}$$

5. Comparative experiments and analyses

To validate our presented model, we make a series of simulation experiments. The simulation platform can effectively simulate the networked multiagent system with dynamic topology; in the simulated system, each agent can sense the dynamic change of its neighboring network structure, thus the negotiation of agent with its context can accord with the real time situation of the network. In the experiments, the task is to collect some types of data from agents, and send the collected data to the monitor; finally, the monitor makes processing on the collected data. In our experiments, there are many kinds of data which are randomly stored in the network, and the collection of each kind of data needs one kind of resource; each agent has different data and resources.

5.1. Comparison between talent-based task allocation and centrality-based task allocation

Here the difference of task execution time between different models is mainly determined by the communication time, thus in this section the execution time is simply described by communication time, i.e. the execution time is 0 if there is only one agent to execute the task.

5.1.1. Semi-supervised allocation without load balancing

Now we make a series of experiments on the semi-supervised task allocation without considering load balancing; the experiments are divided into two categories, one is in the hierarchical network, and the other is in the arbitrary network. The experimental results are shown in Figs. 7 and 8, where the centrality-based allocation method includes the one in G and the one in *RON* which are described in Section 4.1.

(1) Both in Figs. 7 and 8, we can see that the mean geodesic eccentricities of talent-based method is higher than the ones of centrality-based method in G and *RON*; but there are no obvious difference on occupancy percentages and mean geodesic distances. Therefore, it can conclude that the main difference of the two methods (talent-based one and the centrality-based one) is their taking different mean geodesic eccentricities.

(2) From Fig. 7(iii) and Fig. 8(iii), we can see that the difference of mean geodesic eccentricities between the two methods will decrease with the increasing of task required resources. The potential reason is that the number of allocated agents will increase with the increasing resources, thus the increasing agent number may mitigate the difference of geodesic eccentricities between the two methods.

(3) From Fig. 7(iv) and Fig. 8(iv), we can see that the task execution times in talent-based method are always higher than the ones in centrality-based method, especially when the required resources of tasks increase; thus it concludes that the task execution performance of centrality-based method always outperforms the talent-based one, especially while the required resources of tasks are large.

(4) From Fig. 7(iv), we can see that the method based on centrality in *RON* takes the same effect as the one based on centrality in G; but in Fig. 8(iv), we can see that the method based

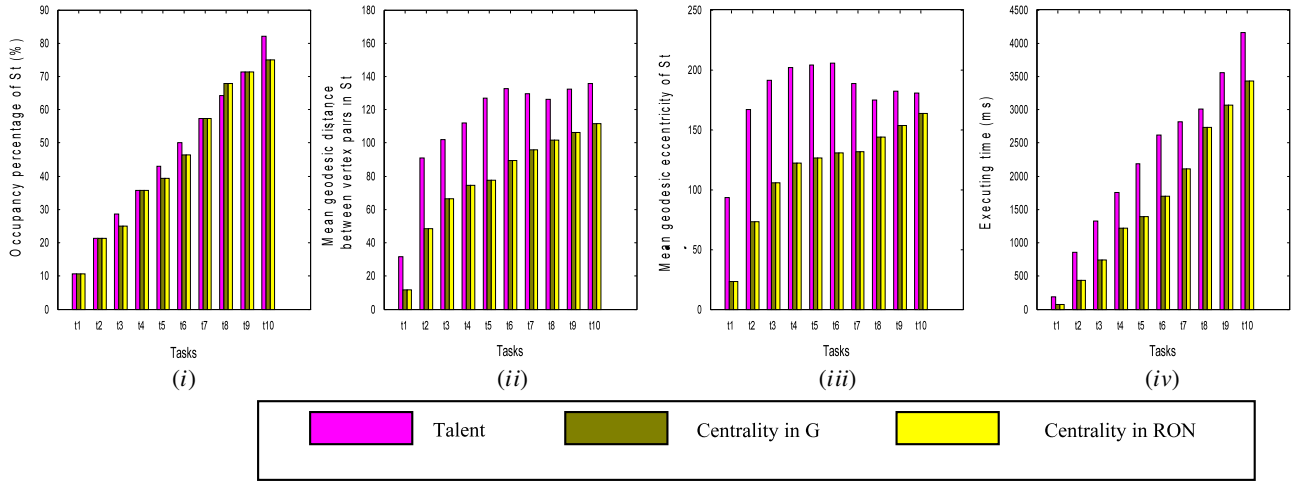


Fig. 7. Semi-supervised allocation without load balancing in hierarchical multiagent network. (i) Occupancy percentage of allocated agents; (ii) Mean geodesic distance between vertex pairs in the allocated agents; (iii) Mean geodesic eccentricity of allocated agents; (iv) Execution time of task.

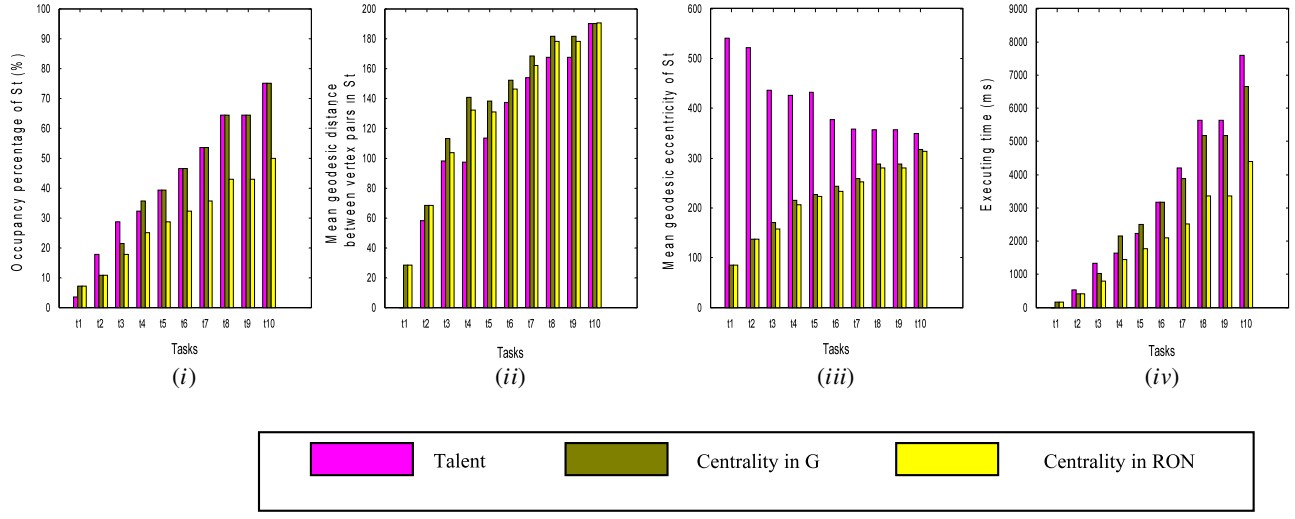


Fig. 8. Semi-supervised allocation without load balancing in arbitrary multiagent network. (i) Occupancy percentage of allocated agents; (ii) Mean geodesic distance between vertex pairs in the allocated agents; (iii) Mean geodesic eccentricity of allocated agents; (iv) Execution time of task.

on centrality in *RON* outperforms the one based on centrality in *G*. The potential reason is: in the arbitrary network structure, the *RON* can make the allocated agents be denser and more central, shown as Fig. 8(i)–(iii); but in the hierarchical structure, the negotiation is constrained much by the structure, thus the one based on *RON* cannot take obvious effects by comparing to the one based on *G*.

5.1.2. Full-supervised allocation without load balancing

Now we make a series of experiments on the full-supervised task allocation without considering load balancing; the experiments are divided into two categories, one is in the hierarchical network, and the other is in the arbitrary network. The experimental results are shown in Figs. 9 and 10.

(1) From Fig. 9(ii)–(iii) and Fig. 10(ii)–(iii), the two allocation methods have different *mean geodesic distance between vertex pairs in* A_t (*GD*) and *mean geodesic eccentricity of* A_t (*GC*), which is different from the result in 5.1.1. Therefore, it concludes that the full-supervised algorithm can get not only different *GDs* but also different *GCs* for the two methods.

(2) From Fig. 9(iii) and Fig. 10(iii), we can see that the difference of mean geodesic eccentricities between the two methods will decrease with the increasing of task required resources. The potential reason is that the number of allocated agents will increase with the increasing resources, thus the increasing agent number may mitigate the difference of geodesic eccentricities between the two methods.

(3) From Fig. 9(iv) and Fig. 10(iv), we can see that the task execution times in talent-based method are always higher than the ones in centrality-based method.

Conclusion. *From Sections 5.1.1 and 5.1.2, we can see that the task allocation based on centrality always outperforms the one based on talent, especially while the tasks are large and the semi-supervised algorithm is used; the reason is that the centrality-based method can result in lower mean geodesic eccentricity of the allocated agents, thus the communication cost will be reduced accordingly. Moreover, the centrality in RON is always equivalent to or outperforms the one in G, since the former kind of centrality also considers the talents of agents.*

5.2. Comparison between talent-based load balancing and centrality-based load balancing

5.2.1. Semi-supervised allocation with load balancing

Now we make a series of experiments on the semi-supervised allocation by *considering*/*not considering* load balancing, the

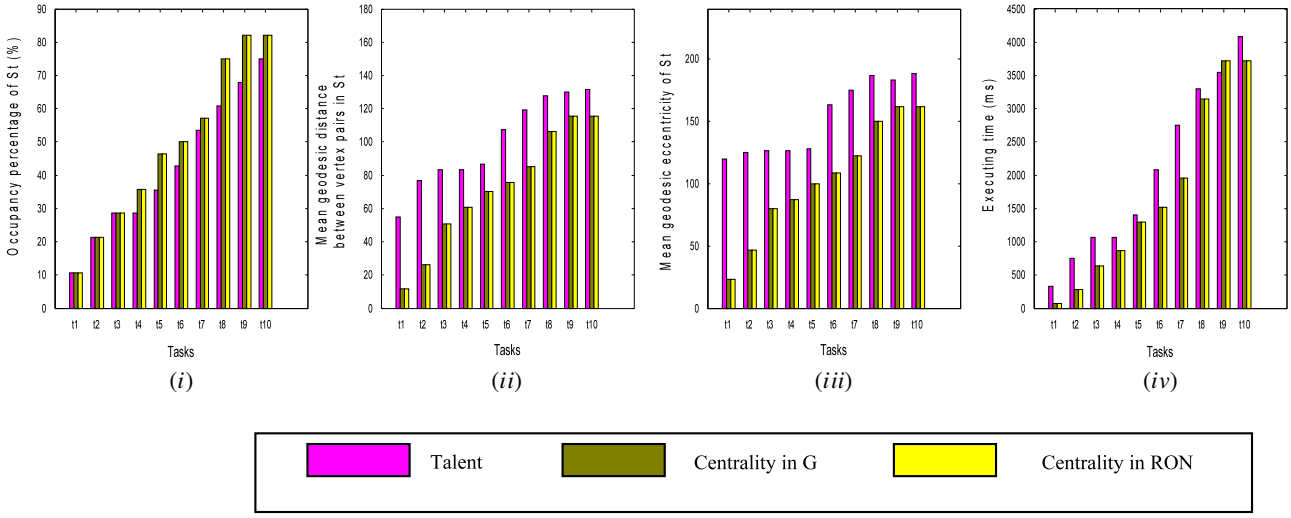


Fig. 9. Full-supervised allocation without load balancing in hierarchical multiagent network. (i). Occupancy percentage of allocated agents; (ii). Mean geodesic distance between vertex pairs in the allocated agents; (iii). Mean geodesic eccentricity of allocated agents; (iv) Execution time of task.

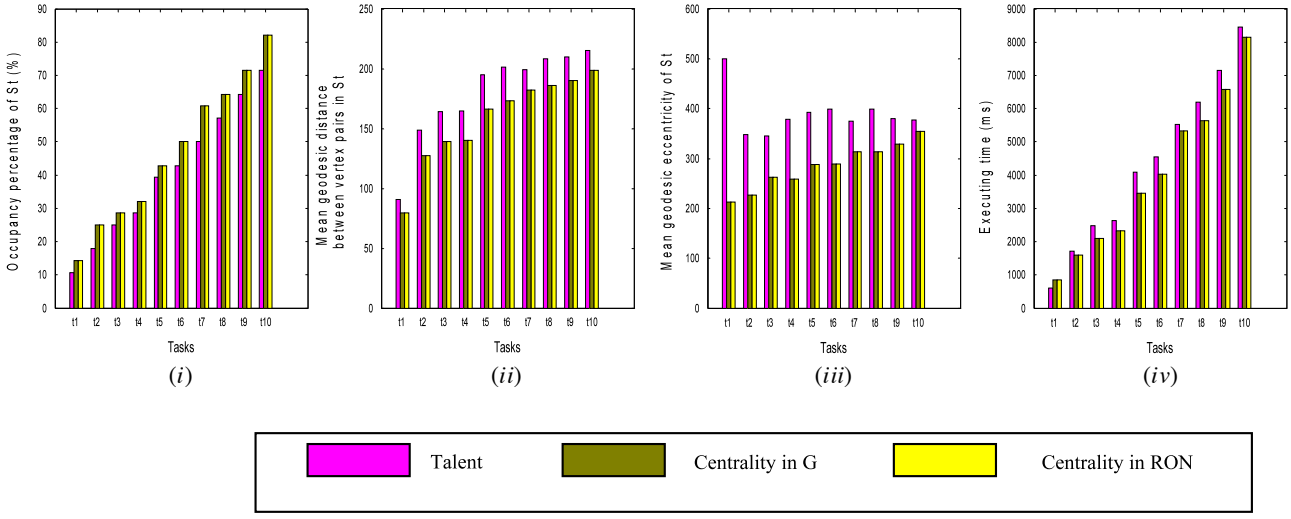


Fig. 10. Full-supervised allocation without load balancing in arbitrary multiagent network. (i). Occupancy percentage of allocated agents; (ii). Mean geodesic distance between vertex pairs in the allocated agents; (iii). Mean geodesic eccentricity of allocated agents; (iv). Execution time of task.

comparative results of total execution time are shown as Fig. 11. In Fig. 11, the x-axis denotes a series of task teams where each task team includes certain tasks, which increases with the required resources from T_1 to T_5; the y-axis denotes the total execution time of a team of tasks. From Fig. 11, we can see that our load balancing mechanism can always take obvious effects which can reduce the total execution times by contrasting to the ones without load balancing; moreover, the effect of load balancing in hierarchical network is better than the one in arbitrary network while the task team is not large.

5.2.2. Full-supervised allocation with load balancing

Now we make a series of experiments on the full-supervised allocation by *considering*/*not considering* load balancing, the comparative results of total execution time are shown as Fig. 12. In Fig. 12, the x-axis denotes a series of task teams where each task team includes certain tasks, which increases with the required resources from T_1 to T_5; the y-axis denotes the total execution time of a team of tasks. From Fig. 12, we can see that our load balancing mechanism can always take obvious effects which can reduce the total execution times by contrasting to the ones without load balancing.

Conclusion. *From Sections 5.2.1 and 5.2.2, we conclude:* (1) *whatever task allocation methods are adopted, the load balancing mechanism can take obvious effects to reduce the total task execution time by comparing to the one without load balancing;* (2) *while load balancing is considered, the talent-based allocation method can generally outperform the centrality-based one, since talent-based method can effectively mitigate the congestion of tasks.*

6. Related work

Our research is related to the task allocation and load balancing of multiagent systems. Generally, the related work can be categorized as follows.

(1) *Task allocation of multiagent systems.* When a task comes to a multiagent system, the first step is to allocate such task to certain agents, which is called task allocation [25,11]. Generally, the task allocation in previous work can be categorized into two kinds: (1) One is *self-owned resource based method*, which is always implemented based on the agents' self-owned resources distribution; the number of allocated tasks on an agent is always directly proportional to its self-owned resources [22,18]. (2) The other is *contextual resource based method*, where the number of

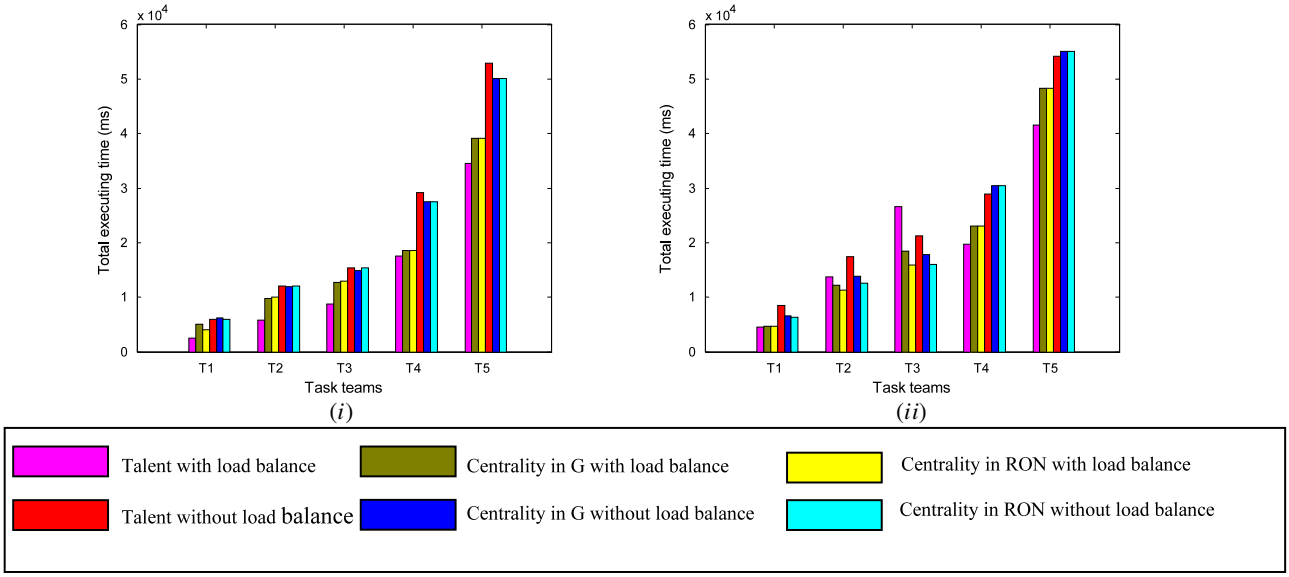


Fig. 11. Semi-supervised allocation with load balancing in multiagent network. (i) Hierarchical multiagent network; (ii) Arbitrary multiagent network.

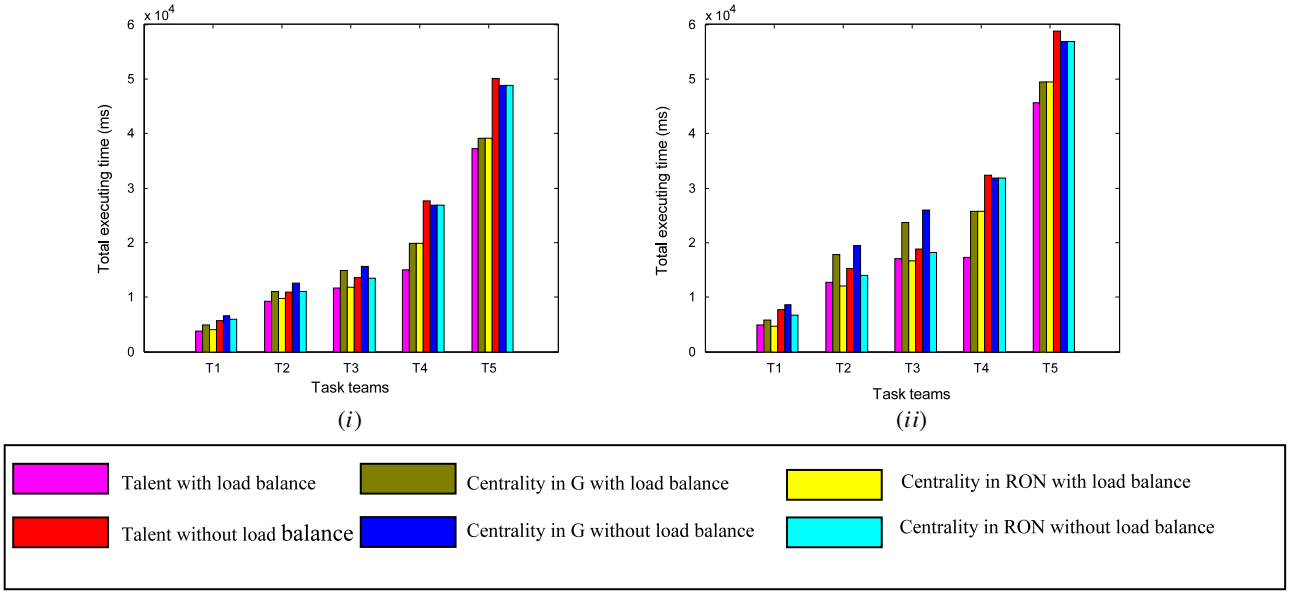


Fig. 12. Full-supervised allocation with load balancing in multiagent network. (i) Hierarchical multiagent network; (ii) Arbitrary multiagent network.

allocated tasks on an agent is directly proportional to not only its self-owned resources but also the resources of its contextual agents, since agents always need to cooperate with other ones within their contexts when they execute tasks [10].

(2) *Workload-based Load balancing of multiagent systems*. An agent may be assigned more tasks, if it possesses more self-owned or contextual resources. However, if too many tasks are crowded on certain agents which have higher resource accessibilities, the agents will have heavy workloads such that the tasks may be delayed and do not get quick responses; therefore, we should let some tasks be switched to other agents with relatively fewer resources but lower workloads, which is called workload-based load balancing. For example, Liu et al. present a macroscopic characterization of agent-based load balancing, and the number and size of teams where agents queue are considered [18]. Schaerf et al. study the adaptive load balancing where the information of global resource distribution should be known to make global optimal resource selections [23]. The workload-based load balancing methods are sufficient to cope with the agents executing unrelated tasks where the communication between agents is few.

(3) *Communication-based load balancing of multiagent systems*. In multiagent systems, the agents always need to exchange messages with each other, thus the communication among agents should take obvious effects in the load balancing [5]. In the related benchmark work in [4], Chow and Kwok investigate the load balancing for distributed multiagent computing, where a novel communication-based load balancing algorithm is proposed by associating a credit value with each agent; the credit of an agent depends on its complete situation, such as its affinity to a machine, its current workload, its communication behavior, etc. The rationale of this credit attribute is to capture the affinity of an agent to the machine in that the intra-machine communication component contributes positively to the credit whereas the reverse is true for the inter-machine communication. The agents with low credits will be selected to migrate to other machines so that the inter-machine communication becomes local communication in

the receiver machine, thereby the total communication overhead can be reduced significantly.

From above, we can see that our locality-based method is some similar to the work in [4], both of which consider the effects of communication among agents. However, there are some differences between them: (1) The algorithm in [4] is to migrate agents among compute hosts so as to implement the *load balancing of compute hosts*, our method is to assign tasks among agents so as to implement *the load balancing of agents*; (2) The communication in [4] is related to the underlying topologies of compute hosts and the affinities of agents to the hosts, but the communication in our model is only related to the topologies of organization network structures among agents; (3) The communication among agents in [4] includes inter-machine communication and intra-machine communication, so the target of load balancing in [4] is to reduce the inter-machine communication, in our model the communication only considers the inter-agent communication (which overlooks the underlying physical networks); (4) There is not a network structure to organize the agents in [4], but in our model all agents are network structure organized so that the interaction locality of agent is very important. Generally, in multiagent systems, the agents are organized in some structures and run on certain underlying computer machines; t*herefore, we can see that the method in* [4] *can be used in any multiagent systems running on the underlying networked machines (i.e., the multiagents themselves may not be organized in network structures), but our model is mainly used in the networked multiagent systems (i.e., the multiagents themselves should be organized in network structures).*

The contributions of this paper. The purpose of this paper is to investigate the task allocation in networked multiagent system, termed locality-sensitive task allocation and load balancing. This paper mainly investigates the comparison between the talent and centrality-based task allocations, and tries to find the effects of such two approaches in different situations. The experiment results show that the centrality-based method can reduce the communication costs for single task more effectively than the talent-based one, but the talent-based method can generally obtain better load balancing performance for parallel tasks than the centrality-based one.

7. Conclusion

In this paper, the task allocation and load balancing in networked multiagent systems are addressed; and a novel locality-sensitive approach is investigated. This paper mainly presents two different models and makes comparison between them: one is the talent-based task allocation and load balancing, which is implemented according to the agents' talents, such as resources; the other one is the centrality-based task allocation and load balancing, which is implemented according to the agents' centralities in network. The experiment results show that the centrality-based method can reduce the communication costs of single task more effectively than the talent-based one; while the talent-based method can generally obtain better load balancing performance of parallel tasks than the centrality-based one. Therefore, the centrality-based allocation method can be applied while the implementation of tasks is serial, and the talent-based allocation method can be applied while the implementation of tasks is parallel.

In our model, the agents' talents are assumed to only differ essentially in their resources; however, in some circumstances the above assumption does not match the peculiarities of real multiagent systems where some agents may have different other talents besides resources [10], such as computable functions, sensing abilities. Obviously, by only extending the definition of talent, our talent-based model can easily be extended into other real situations where agents have different other factors.

Moreover, in this paper, the centrality is measured based on closeness. However, there are various kinds of definitions for centrality. Therefore, in the future, we will investigate the effects of different centrality measures on the locality-sensitive task allocation.

Acknowledgments

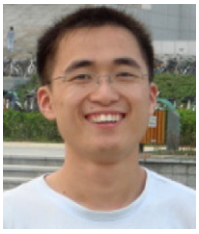
This research was supported by the National Natural Science Foundation of China (No. 60803060), the Specialized Research Fund for the Doctoral Program of Higher Education of State Education Ministry of China (No. 200802861077, No. 20090092110048), the General Program of Humanities and Social Sciences in University of State Education Ministry of China (No. 10YJCZH044), the Program for New Century Excellent Talents in University of State Education Ministry of China (No. NCET-09-0289), the National High Technology Research and Development Program of China (863 Program) (No. 2009AA01Z118), the Open Research Fund from Key Laboratory of Computer Network and Information Integration in Southeast University, State Education Ministry of China (No. K93-9-2010-16), the Excellent Young Teachers Program of Southeast University (No. 4050181013), and the State Key Laboratory for Manufacturing Systems Engineering (Xi'an Jiaotong University) (No. 2010001).

References

[1] Sherief Abdallah, Victor Lesser, Multiagent reinforcement learning and self-organization in a network of agents, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS07), Honolulu, Hawaii, May, 2007, pp. 172–179.
[2] Baruch Awerbuch, David Peleg, Locality-sensitive resource allocation. Technical Report CS90-27, Weizmann Institute, November 1990.
[3] Stephen P. Borgatti, Centrality and network flow, Social Networks 27 (1) (2005) 55–71.
[4] Ka-Po Chow, Yu-Kwong Kwok, On load balancing for distributed multiagent computing, IEEE Transactions on Parallel and Distributed Systems 13 (8) (2002) 787–801.
[5] Ka-Po Chow, Yu-Kwong Kwok, Design and evaluation of a novel communication-based approach to multi-agent load balancing, in: Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems 2000, PDCS 2000, vol. 1, Las Vegas, Nevada, USA, November 2000, pp. 329–334.
[6] Sagar Dhakal, Majeed M. Hayat, Jorge E. Pezoa, Cundong Yang, David A. Bader, Dynamic load balancing in distributed systems in the presence of delays: a regeneration-theory approach, IEEE Transactions on Parallel and Distributed Systems 18 (4) (2007) 485–497.
[7] Peter Sheridan Dodds, Duncan J. Watts, Charles F. Sabel, Information exchange and the robustness of organizational networks, Proceedings of the National Academy of Sciences 100 (21) (2003) 12516–12521.
[8] Yichuan Jiang, Extracting social laws from unilateral binary constraint relation topologies in multiagent systems, Expert Systems with Applications 35 (4) (2008) 2004–2012.
[9] Yichuan Jiang, Concurrent collective strategy diffusion of multiagents: the spatial model and case study, IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews 39 (4) (2009) 448–458.
[10] Yichuan Jiang, Jiuchuan Jiang, Contextual resource negotiation-based task allocation and load balancing in complex software systems, IEEE Transactions on Parallel and Distributed Systems 20 (5) (2009) 641–653.
[11] Yi-Chuan Jiang, J.C. Jiang, A multi-agent coordination model for the variation of underlying network topology, Expert Systems with Applications 29 (2) (2005) 372–382.
[12] Yichuan Jiang, Jiuchuan Jiang, Toru Ishida, Compatibility between the local and social performances of multi-agent societies, Expert Systems with Applications 36 (3-Part 1) (2009) 4443–4450.
[13] Guo Jiani, L.N. Bhuyan, Load balancing in a cluster-based web server for multimedia applications, IEEE Transactions on Parallel and Distributed Systems 17 (11) (2006) 1321–1334.
[14] Joseph S. Kong, Nima Sarshar, Vwani P. Roychowdhury, Experience versus talent shapes the structure of the web, Proceedings of the National Academy of Sciences 105 (37) (2008) 13724–13729.
[15] Jin-Shyan Lee, Pau-Lo Hsu, Implementation of remote hierarchical supervision system using Petri Nets and agent technology, IEEE Transactions on System, Man and Cybernetics, Part C; Applications and Reviews 37 (1) (2007) 77–85.

[16] Yann Le Quéré, Marc Sevaux, Christian Tahon, Damien Trentesaux, Reactive scheduling of complex system maintenance in a cooperative environment with communication times, IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews 33 (2) (2003) 225–234.
[17] Donghui Lin, Huanye Sheng, Toru Ishida, Interorganizational workflow execution based on process agents and ECA rules, IEICE Transactions on Information and Systems E90-D (9) (2007) 1335–1342.
[18] Jiming Liu, Xiaolong Jin, Yuanshi Wang, Agent-based load balancing on homogeneous minigrids: macroscopic modeling and characterization, IEEE Transactions on Parallel and Distributed Systems 16 (7) (2005) 586–598.
[19] Kian Hsiang Low, Wee Kheng Leow, Marcelo H. Ang Jr., Autonomic mobile sensor network with self-coordinated task allocation and execution, IEEE Transactions on System, Man, and Cybernetics-Part C: Applications and Reviews 36 (3) (2006) 315–327.
[20] B.R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in C++, John Wiley & Sons, New York, 1999.
[21] Weixiong Rao, Lei Chen, Ada Wai-Chee Fu, Guoren Wang, Optimal resource placement in structured peer-to-peer networks, IEEE Transactions on Parallel and Distributed Systems 21 (7) (2010) 1011–1026.
[22] Sarit Kraus, Tatjana Plotkin, Algorithms of distributed task allocation for cooperative agents, Theoretical Computer Science 242 (1–2) (2000) 1–27.
[23] Andrea Schaerf, Yoav Shoham, Moshe Tennenholtz, Adaptive load balancing: a study in multi-agent learning, Journal of Artificial Intelligence Research 2 (1995) 475–500.
[24] Kiam Tian Seow, Kwang Mong Sim, S.Y.C. Kwek, Coalition formation for resource coallocation using BDI assignment agents, IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews 37 (4) (2007) 682–693.
[25] Onn Shehory, Sarit Kraus, Methods for task allocation via agent coalition formation, Artificial Intelligence 101 (1–2) (1998) 165–200.

Yichuan Jiang received the PhD degree in computer science from Fudan University, China, in 2005. He is currently a professor in the Lab of Complex Systems and Social Computing, School of Computer Science and Engineering, Southeast University, Nanjing, China. His main research interests include multiagent systems, complex distributed systems and social computing. He was awarded the New Century Excellent Talents in University of State Education Ministry of China, the Best Paper Award from PRIMA2006, and the Nomination Award for the National Excellent Doctoral Dissertation of China, the Young Distinguished Professor of Southeast University. He is also the recipient of the HwaYing Young Scholar Award in 2008 from The HwaYing Education and Culture Foundation. He has published more than 60 scientific articles in refereed journals and conference proceedings, such as IEEE TPDS, IEEE TSMC-A, IEEE TSMC-C, IJCAI and AAMAS. He is a member of IEEE and the IEEE Computer Society, a member of ACM, a senior member of China Computer Federation, a senior member of Chinese Institute of Electronics, and a member of editorial board of Chinese Journal of Computers.



Zhaofeng Li received the B.E. degree in information engineering from the School of Information Science and Engineering, Southeast University, Nanjing, China, in 2009. He is currently a postgraduate student in the Lab of Complex Systems and Social Computing, School of Computer Science and Engineering, Southeast University, Nanjing, China. His main research interests include complex multiagent systems and distributed computing.