The Rich Get Richer: Preferential Attachment in the Task Allocation of Cooperative Networked Multiagent Systems With Resource Caching

Yichuan Jiang, *Member, IEEE*, and Zhichuan Huang

***Abstract*—In networked multiagent systems (NMASs) with resource caching, resource replicas are cached in favor of the agents who accessed such resources most recently and frequently. Task execution in NMASs is described through agents' operations when accessing necessary resources distributed in the networks, and thus, agents with richer experiences executing tasks will have higher access to resources. To optimize tasks' resource access time, we investigate two types of preferential attachments in the task allocation of NMASs with resource caching: history and present preferential attachments, in which an agent has higher access to a resource if that agent has richer history (or present) accessing experiences for that resource. Therefore, agents that were (or are) heavily burdened by tasks may have certain preferential rights to new tasks in the future. Our experiments found that preferential attachment in task allocation can effectively reduce tasks' execution time, particularly when the network context is considered and the number of tasks is high. In addition, we discovered two interesting phenomena: 1) Compromise between preferential attachment and load balancing can achieve better performance than single preferential attachment when there are too many tasks waiting, and 2) the integration of history and present preferential attachments can outperform either history or present preferential attachment alone in task allocation.**

***Index Terms*—Distributed systems, load balancing, networked multiagent systems (NMASs), preferential attachment, resource caching, task allocation.**

Manuscript received September 19, 2010; revised May 23, 2011 and September 18, 2011; accepted December 27, 2011. Date of publication March 2, 2012; date of current version August 15, 2012. This work was supported in part by the National Natural Science Foundation of China under Grants 61170164 and 60803060, the Specialized Research Fund for the Doctoral Program of Higher Education of State Education Ministry of China under Grants 200802861077 and 20090092110048, the General Program of Humanities and Social Sciences in University of State Education Ministry of China under Grant 10YJCZH044, the Program for New Century Excellent Talents in University of State Education Ministry of China under Grant NCET-09-0289, and the Special Fund for Fast Sharing of Science Paper in Net Era by Center of Science and Technology Development of State Education Ministry of China under Grant 20110092110053. This paper was recommended by Associate Editor K. T. Seow.

Y. Jiang is with the Key Laboratory of Computer Network and Information Integration of State Education Ministry, School of Computer Science and Engineering, Southeast University, Nanjing 211189, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China (e-mail: jiangyichuan@yahoo.com.cn).

Z. Huang is with the Laboratory for Complex Systems and Social Computing, School of Computer Science and Engineering, Southeast University, Nanjing 211189, China, and also with the State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an 710054, China (e-mail: onlyhzc@gmail.com)

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2012.2186439

I. Introduction

To MEET the increasing demand of large-scale cooperative computational problems, cooperative distributed systems (CDSs) have been investigated [1]–[4]. There are many practical CDSs, such as grids [1], [3], peer-to-peer (P2P) systems [4], and ad hoc networks [2]. In general, many CDSs have the following characteristics.

1) *Nodes are autonomous and cooperative*. Each node behaves autonomously by considering the surrounding situations [5]; the nodes can contribute their idle resources and cooperatively work together to accomplish tasks [6].
2) *Nodes are interconnected through networks, and interactions are local and constrained by network* structures. Network structures can cope with large-scale situations because each node only needs to know its surrounding situations [6], [7]. Moreover, network structures can save nodes' energies because each node only communicates with its neighbors and remote communication can be implemented by forwarding among nodes [2].
3) *Resources are distributed within the networks, and access to resources is crucial to the system performance*. Many CDSs aim to achieve resource sharing, and thus, communication time for accessing resources is crucial [1], [3], [4].
4) *Resource caching can be used to improve the performance of resource access*. Some resources are replicated at certain places in the networks so that the access to those resources is easier [2], [8], [9].

CDSs can be viewed as networked multiagent systems (NMASs) in which agents represent the autonomous nodes and interaction relations represent interconnections among nodes [6]. The concept of NMASs enables users to represent CDSs at an abstract level without needing to worry about the particulars of the target system [6], [7], [10], [11]. An NMAS can be depicted as a graph, $G = \langle A, E\rangle$, consisting of vertices (agents A) and edges (interactions E); some resources are placed within the network and can be accessed by agents for the execution of tasks [12].

Task execution in multiagent systems can be described by the agents' operations when accessing required resources; thus, task allocation is often implemented based on the accessibility of required resources [1], [13]–[16]. In previous work, load balancing was necessary for the allocation of multiple tasks to reduce tasks' waiting time at agents so that tasks could be switched from heavy-burdened agents to light-burdened ones

[1]. If there are too many tasks queuing for an agent, the probability of the agent being assigned new tasks is reduced. Therefore, load balancing in previous work is in accordance with the adage "winner does not take all."

However, previous approaches to task allocation and load balancing do not match the peculiarities of real NMASs with resource caching, as described in the following points.

1) In NMASs, resource caching is implemented to improve resource access to enable the agents to execute allocated tasks [2], [9]. If an agent has richer experiences of executing tasks (i.e., the agent is heavily burdened), it is more probable that resource replicas will be placed to ease the agent's access. Therefore, the agent's resource access time in future task execution can be reduced [2], [8], [9]. Although previous load balancing methods may reduce the waiting time of a task by switching it to a light-burdened agent, it will take more time to access resources from the light-burdened agent than from the heavy-burdened agent. Considering the importance of communication time for accessing required resources in the NMASs, it may be better to simply assign the task to the heavy-burdened one.
2) Previous task allocation and load balancing methods based on resources are often developed on the assumption that the resources necessary to the tasks and resources available from the agents are known; thus, each time that the task allocation is implemented, the accurate information of resource distribution in the system should be obtained timely [1], [12]. However, in NMASs with resource caching, resources may be dynamically replicated and distributed within the networks, particularly while many tasks are executed [2]–[4], and it may be difficult to acquire accurate resource information in a timely fashion. Although there are some related studies on task allocation with uncertain information [17]–[21], such methods may bring about heavy costs for agents' computing and communication, influencing the overall performance of the systems, particularly when the number of tasks is high.

To deal with the aforementioned problems, we propose the following novel idea of task allocation based on preferential attachment: Experienced agents receive more new tasks than less experienced agents. Obviously, this task allocation method accords with the social cliché "the rich get richer" [22], [23]. In this paper, we investigate two types of preferential attachments in task allocation: history and present preferential attachments. Our basic reason is as follows: Due to resource caching, an agent has higher future access to a resource if the agent has richer history (or present) access experiences for the resource; therefore, the agents that were (or are) heavily burdened with tasks may have preferential rights to receive new tasks in the future, which can reduce the agents' communication time to access resources. Therefore, our approach can be used in the NMASs, in which communication time for the required resource is crucial to executing tasks.

Moreover, because our task allocation approach is implemented *by simply relying on agents' experiences of executing tasks*, it does not need to fully understand accurate information about available resources in real time and can avoid bringing about heavy costs for agents' computing and communication. Therefore, our approach is better suited than the previous work for large-scale and dynamic NMASs.

The rest of this paper is organized as follows. In Section II, we compare our work with the previous work. In Section III, we describe NMASs with resource caching. In Section IV, we formalize the optimization problem of task allocation. In Section V, we propose a task allocation model based on preferential attachment. In Section VI, we provide experimental results to validate our proposed model. Finally, we discuss and conclude this paper in Section VII.

II. Related Work

Our research is related to resource-based task allocation approaches, centralized and distributed task allocation approaches, task allocation with uncertain information, and load balancing in task allocation. Generally, related work can be categorized as follows.

1) ***Resource-based task allocation approaches***.
   The main goal of task allocation is to maximize the overall performance of the system and fulfill tasks as quickly as possible [24], [25]. Many related works also aim to optimize tasks' execution time [13], [14]. Without loss of generality, task execution can be described through the agents' operations when accessing required resources [1], [25]; therefore, resource accessibility may influence tasks' execution time. Many related works were implemented based on two types of resources: 1) a self-owned resource-based approach implemented based on the agents' self-owned resource status [1], [16] and 2) a contextual resource-based approach implemented based on not only the agents' self-owned resource status but also their contextual resource status because agents may cooperate with others within their contexts when they execute tasks [15]. Generally, previous works based on resource accessibility assume that accurate resource information can be known.
2) ***Centralized and distributed approaches.***
   On the other hand, traditional task allocation works can be categorized into centralized [26] and distributed approaches [27] according to their allocation control mechanisms. In the centralized approach, there is a centralized controller that implements task allocation. For example, Fjuita and Lesser [28] present deadline-based multiagent task decomposition and scheduling in an environment in which one master agent is responsible for task allocation and needs to know the information of the whole system. In the distributed approach, a centralized controller is not needed. For example, Shehory and Kraus [29] present methods for task allocation via agent coalition formation without a central authority. Bo An *et al.* [13], [14] present the benchmark optimization method for multiresource negotiation in task allocation, which utilizes a time-dependent negotiation strategy in which the reserve price of each resource is dynamically determined. Another typical distributed method is the contract net protocol [30],

a well-known task sharing protocol in which each agent in a network can be a manager or a contractor at different times or for different tasks [31]. However, managers in this method also need to know the information of negotiated agents. In summary, agent status information needs to be acquired in a timely manner for these related works.

3) ***Task allocation with uncertain information.***

There are some related studies considering task allocation with uncertain and incomplete information. In previous approaches, social or economics techniques are used, such as negotiation, bidding/auction, trust, game theory, etc. For example, Kraus *et al.* [17] present a distributed approach in which a protocol is developed to enable agents to negotiate and form coalitions to execute tasks with uncertain heterogeneous information. Ramchurn *et al.* [18] present trust-based mechanisms of task allocation in the presence of execution uncertainty, which take into account the trust between agents when allocating tasks. Mataric *et al.* [19] use the bidding/auction mechanism to perform task allocation in uncertain environments. Game theory is used in the task allocation of some heterogeneous distributed systems in which the complete information of agent peers is unknown. For example, Grosu and Chronopoulos [21] present a game-theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. Moreover, in multirobot systems, a dynamic task allocation mechanism is investigated, which allows agents to change their behavior in response to environmental changes or other agents' actions to improve overall system performance [20]. Such a dynamic allocation mechanism needs agents to have sensing and decision-making abilities.

In summary, the aforementioned approaches may bring about heavy costs for agents' computing and communication, which may influence overall system performance, particularly when the number of tasks is high.

4) ***Load balancing in task allocation.***

If too many tasks are allocated to certain agents, the tasks may be delayed and will not receive quick responses. To minimize the amount of time that tasks remain with an agent, tasks may be switched to other agents with lower task loads, which is called *load balancing*. Liu *et al.* [1] present a macroscopic characterization of agent-based load balancing, in which complete information about the number and size of task teams queuing for agents is necessary. Chow and Kwok [32] investigate load balancing for distributed multiagent computing, in which 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 current situation, such as its affinity to a machine, its current workload, its communication behavior, etc. Schaerf *et al.* [12] study adaptive load balancing, in which information on global resource distribution must be known to make global optimal resource selections. Dhakal *et al.* [33] present a regeneration-theory approach to dynamic load balancing in distributed systems in the presence of delays, in which heterogeneity in the processing rates of the nodes is taken into account.

From above, we can see that load balancing is an important idea for the allocation of multiple tasks, where the number of tasks queuing for an agent is the determinative factor of the agent's rights in future task allocation. If there are too many tasks queuing for an agent, the probability of the agent getting new tasks will be reduced [1]. Therefore, the previous load balancing method of task allocation accords with the adage "winner does not take all" [34].

5) ***Our contribution.***

In comparison with the related work, our work makes the following contributions.

a) Previous load balancing methods based on the "winner does not take all" theory can effectively reduce the waiting time of tasks at agents [1]. In contrast to previous work, our "rich get richer" model can effectively reduce the communication time of agents attempting to access resources within the network. In fact, we combine "rich get richer" and "winner does not take all" to implement a compromise between preferential attachment and load balancing, which can achieve better performance than single preferential attachment while there are too many waiting tasks.

b) Previous resource-based task allocation and load balancing methods are often premised on the assumption that the resources needed by the tasks and resources available from the agents are known; therefore, each time that the task allocation is implemented, accurate resource distribution information needs to be acquired timely from the system [6]. In contrast with the previous work, our model is implemented by simply relying on agents' experiences of executing tasks and frees task allocation from knowing the status of available resources. Our model applies well to large dynamic NMASs, in which it is difficult to obtain accurate resource status information timely.

c) Although there are some related studies on the task allocation with uncertain information, they may bring about heavy costs to the agents' computing and communication, which may influence overall system performance, particularly when the number of tasks is high. Our approach avoids bringing about heavy costs for agents' computing and communication because it is implemented by simply relying on agents' experiences of executing tasks.

III. NMASs With Resource Caching

A. Resource Caching in Multiagent Networks

In previous benchmark work on resource caching, two typical methods are used: One is plain caching, which means that resource replicas are placed at the requesting agents [35], and the other is intermediate caching, which means that resource replicas are placed at intermediate agents (the agents between a requesting agent and the agent that holds the requested resource) [2], [8], [9]. Generally, intermediate caching is more beneficial for the performance of systems in which the allocated agents are not fixed; moreover, with the intermediate caching method, the number of replicas within the network can be more effectively reduced than with the method that places the replicas at the requesting agents.

Resource caching is not our focus in this paper. Therefore, without loss of generality, we present a simple resource caching mechanism based on abstracting from some related intermediate caching methods [2], [8], [9]. In resource caching, cache locations can be dynamically adapted according to the frequency of resource access. We also describe the eclipse mechanism of resource replicas and make the seldom-accessed resource replicas eclipse step by step, which results in saving of agent storage.

1) Resource Caching Mechanism: While an NMAS is initially set up, the locality of resource r_i is called r_i's *original inhabitation locality*, denoted as OL_{ri}. After the system runs, r_i may be replicated at a place in the network, referred to as r_i's *caching locality*, CL_{ri}.

Obviously, the original inhabitation locality of a resource is always fixed, but a resource's caching locality can be changed. *Let two localities in the network be L_i and L_j. Let the shortest path betweenL_i and L_j be $\{L_i, L_{i+1}, \ldots, L_{j-2}, L_{j-1}, L_j\}$. If a resource changes its caching locality from L_j to L_{j-n}* $(1 \leq n \leq j-i)$, *we can say that the resource replica migrates from L_j to L_i with n hops; the new locality after migration is represented as $L^n_{L_j \to L_i}$.*

Now, we propose a simple resource caching mechanism as Algorithm 1, where T is the set of tasks.

Algorithm 1. Resource caching in task execution.

```
1) $\forall t \in T$:
  1.1) Allocate the principal agent for $t$, $a_t$.
  1.2) $\forall r \in R_t$:
    **If** the current locality of accessed resource $r$ is its
       original inhabitation locality, $OL_r$:
         1.2.1) Produce a replica of $r$, which is represented
            by $cr$.
         1.2.2) Migrate $cr$ to $L^n_{OL_r \to L_{a_t}}$.
    **else Migrate** $r$ **to** $L^n_{CL_r \to L_{a_t}}$.
2) **End**.
```

From Algorithm 1, when a resource in the original inhabitation locality is accessed, we should produce a replica of the resource and move the replica toward the allocated agent with a series of hops. When the resource replica in the caching locality is accessed, we will only need to move the replica toward the allocated agent with a series of hops.

Definition 1: Compactness degree of an agent set. *Given a set of agents*, A, *whose compactness degree is the inverse of the mean length of the shortest paths between each pair of agents, shown as*

$$CD_A = 1/\left(\left(\sum_{a_i, a_j \in A} d_{ij}\right) / (|A| \cdot (|A|+1))\right) \quad (1)$$

where d_{ij} denotes the length of the shortest path between a_i and a_j and $|A|$ denotes the number of agents in A. Obviously, the higher CD_A is, the more compact the agents in A are.

Theorem 1: Given two agent sets in the multiagent network $G = \langle A, E\rangle$, $A_1, A_2 \subseteq A$, $|A_1| = |A_2|$; all agents inA_1 and A_2 will access resource r. The probability of producing a replica of r by A_1 is $P_{A1}(r)$, and the probability of producing a replica of r byA_2 is $P_{A2}(r)$. We have the following: $CD_{A1} > CD_{A2} \Rightarrow P_{A1}(r) \leq P_{A2}(r)$.

Proof: We use the inductive method to prove Theorem 1.

1) While $|A_1| = |A_2| = 2$, the agent first accessing r in A_i is a_{i1}, and the second agent accessing r in A_i is a_{i2}. Because the agents in A_1 are more compact than the agents in A_2, the probability that a_{12} accesses the resource replica produced by a_{11} is higher than the probability that a_{22} accesses the resource replica produced by a_{21}. Thus, the probability that a_{12} produces a new resource replica is less than the probability that a_{22} produces a new resource replica. Therefore, we have $P_{A1}(r) \leq P_{A2}(r)$.
2) While $|A_1| = |A_2| = k > 2$, we assume $P_{A1}(r) \leq P_{A2}(r)$.
3) While $|A_1| = |A_2| = k+1$, the agent jth-*ly* accessing r in A_i is $a_{i,j}$. According to Step 2, $P_{A_1-\{a_{1,k+1}\}}(r) \leq P_{A_2-\{a_{2,k+1}\}}(r)$. Because $CD_{A1} > CD_{A2}$, the probability that $a_{1,k+1}$ accesses the resource replica produced by any agent in $(A_1 - \{a_{1,k+1}\})$ is higher than the probability that $a_{2,k+1}$ accesses the resource replica produced by any agent in $(A_2 - \{a_{2,k+1}\})$. Thus, the probability that $a_{1,k+1}$ produces a new resource replica is less than the probability that $a_{2,k+1}$ produces a new resource replica. Therefore, we have $P_{A1}(r) \leq P_{A2}(r)$.

□

From Theorem 1 and Definition 1, we conclude that the set with more compact agents produces fewer resource replicas. Therefore, we should seek the set with more compact agents to accomplish the task.

2) Eclipse of Resource Caching: To avoid the congestion of redundant resource replicas in the multiagent network, we can let some longtime unused resource replicas be terminated. This process is called the *eclipse of resource caching*.

While a cached resource replica has not been accessed for long time, we let it go back toward its original inhabitation locality step by step. When it arrives at its original inhabitation locality, it can be terminated. We can set a time period T; after every T time, all cached resource replicas in the multiagent network will withdraw to their respective original inhabitation localities with one hop. The cached resource replicas remaining in the network are those that are accessed by agents most recently and frequently. Idle resource replicas will eclipse step by step.

Let $\boldsymbol{N}(i)$ be the maximum number of resource replicas in the system at time i, N_t be the number of tasks arrived at a time unit, N_r be the mean number of resources needed by one task, T be the eclipse time, $|R|$ be the mean number of resources that an agent owns, and $N_e(i)$ be the number of replicas eclipsed at time i.

Then, we formalize $\boldsymbol{N}(i+1)$ as

$$N(i+1) = N(i) + N_t^* (N_r - |R|) - N_e(i) \tag{2}$$

Finally, after a long enough time, we have

$$\lim_{i\to\infty} N(i+1) = \lim_{i\to\infty} (N(i) + N_t * N_r - |R| - N_e(i)) \tag{3}$$

Because the limits of $N(i+1)$ and $N(i)$ are identical and due to the eclipse mechanism

$$\lim_{i\to\infty} N_e(i) = \lim_{i\to\infty} N(i)/T$$

Therefore, we have

$$\lim_{i\to\infty} N(i) = N_i * (N_r - |R|) * T \tag{4}$$

Therefore, we can adjust eclipse time T to control the maximum number of resource replicas in the system.

B. Resource Search in Multiagent Networks

When an allocated agent (the *principal agent*) lacks the necessary resources to implement an allocated task, it may negotiate with other agents in the network; if other agents have the required resources (we call agents that provide resources to the principal agent *assistant agents*), then the principal agent and assistant agents will cooperate to implement the task. In [15], we applied the concept of contextual negotiation into a multiagent network and presented a novel contextual resource search model. Now, we briefly introduce it here. Interested readers can refer to our previous work in [15] for more details.

Let a_i be the principal agent for task t and the resources owned by a_i be R_{ai}. If the set of required resources for t is R_t, then the set of resources lacking for a_i to implement t is as follows: $\overline{R_{a_i}^t} = R_t - R_{a_i}$.

If 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 required by a_i to implement t, then the set of lacking resources of a_i to implement t will be deduced as follows: $\overline{R_{a_i}^t} = \overline{R_{a_i}^t} - R_{a_j}$.

Agent a_i will negotiate with other agents from near to far in the network until all required resources are satisfied, shown as Algorithm 2. After Algorithm 2 is completed, a set of agents for task t will be selected, denoted as A_t (which includes a_i and the assistant agents). Then, the agents in A_t will cooperatively accomplish task t.

Algorithm 2. Resource search of agent a_i in network. /* a_i is the principal agent, and 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 $(Q, a_i)$;
4) **Set** the tag of $a_i$ to 1;
5) $b = 0$;
6) $A_t = \{a_i\}$;
7) $\overline{R_{a_i}^t} = R_t - R_{ai}$;
8) **If** $\overline{R_{a_i}^t} == \{\}$ **then** $b = 1$;
9) **While** ((!EmptyQueue $(Q)$) and $(b == 0)$) **do**:
    9.1) $aout =$ **Out** Queue$(Q)$;
    9.2) $R' = \overline{R_{a_i}^t} - R_{aout}$;
    9.3) **If** $R' \neq \overline{R_{a_i}^t}$ **then**:
        9.3.1) $\overline{R_{a_i}^t} = \overline{R_{a_i}^t} - R_{aout}$;
        9.3.2) $A_t = A_t \cup \{aout\}$;
    9.4) **If** $\overline{R_{a_i}^t} == \{\}$ **then** $b = 1$;
    9.5) $\forall\, alocal \in L_{aout}$:
/* $L_{aout}$ is the set of neighbors of $aout$ */
        **if** the tag of $alocal$ is 0, **then**:
            9.5.1) **Insert** Queue $(Q, alocal)$;
            9.5.2) **Set** the tag of $alocal$ to 1;
10) **If** $(b == 1)$ **then Return** $(A_t)$
else Return (False);
11) **End**.
```

From Algorithm 2, the principal agent will search its cooperating agents for resources from near to far locations to obtain the most compact set of allocated agents. According to Theorem 1, task execution by such allocated agents can produce the smallest number of new resource replicas in the network.

IV. Formalization of Task Allocation

A. Optimization Problem of Task Allocation

The main goal of task allocation is to maximize the overall performance of the system and fulfill the tasks as quickly as possible [24], [25]. Thus, one of the objectives of task allocation is to minimize the execution time of each task [1], [21], [32]. Because task allocation can be described as an optimization problem [14], [36], we now formalize our objective to optimize task allocation by extending the approach in [1] which is implemented based on the optimization of the length of time that tasks wait at agents.

When a task arrives in the system, the first step is to assign the task to an agent (principal agent), which takes charge of execution of the task. Next, the principal agent will seek assistant agents for resources. Finally, the principal agent and assistant agents will cooperatively execute the task; therefore, communication time between the principal and assistant agents is very important [15], [32]. *Therefore, to reduce the execution time of the task, we can reduce the utilities of two factors: the* ***waiting time of the task at agents and the communication time*** *between the principal and assistant agents (i.e., the communication time for accessing required resources within the network).*[1]

When a task t arrives, the system selects its principal agent, denoted as a_t. The set of principal and assistant agents of t is denoted as A_t. Without loss of generality, it can be assumed that a resource can be accessed by only one task at the same

[1]In fact, task execution time also includes the processing time at agents. Because this paper is concerned with the cooperation between agents in task execution, now, processing times at all agents are assumed to be the same and can be neglected.

time. Thus, if the principal and assistant agents are executing a task, a new task would have to wait until all required resources are available. Let E_t be the execution time of task t, W_{tj} be the waiting time of task t for resources at agent a_j, and $C(a_t, a_j)$ be the communication time between a_t and a_j. The purpose of the task allocation is to select the agent set A_t to minimize the execution time of t

$$E_t = \sum_{a_j \in A_t} W_{tj} + \sum_{a_j \in A_t, a_j \neq a_t} C(a_t, a_j) \tag{5}$$

under the constraints that agents' resources are limited and each resource can be accessed by only one task at a time. Therefore, our task allocation goal can be described as the reduction of the utilities of W_{tj} and $C(a_t, a_j)$.

B. *Idealized Approach and Our Idea*

The naive way to select the optimal A_t can be implemented by exhaustive-trial method. In this approach, we can perform an exhaustive trial by letting each agent act as the principal one and select the one with the minimum execution time E_t. Obviously, although the exhaustive-trial method can return the optimal task allocation result regarding execution time, it has the following drawbacks: 1) It needs a centralized controller to implement exhaustive trials for all agents, which will *waste much allocation time* and is not applicable to large NMASs, and 2) it needs the centralized controller to know all information about the system timely, which is also not applicable to dynamic NMASs. Therefore, a centralized scheduler based on the exhaustive-trial method is not feasible for the handling of the practical task allocation problem in large-scale and dynamic NMASs.

To address the aforementioned problem, this paper presents a novel idea of task allocation *by simply relying on agents' experiences of executing tasks*, which does not need to know all the accurate information of available resources in real time. Our basic idea is this: Due to resource caching, an agent has higher future access to a resource if the agent has richer history (or present) accessing experiences for the resource; thus, agents that were (or are) heavily burdened with tasks may have preferential rights to receive new tasks in the future. *The advantage of our allocation idea is that it can reduce the time for resource access, i.e., reduce the communication time in* (5).

We also apply load balancing in task allocation when the benefit of preferential attachment (reduction in communication time) is less than the loss brought by the waiting time of queuing tasks. *Now, load balancing on the base of preferential attachment in task allocation can reduce both waiting time and communication time in* (5).

V. PREFERENTIAL ATTACHMENT-BASED TASK ALLOCATION

A. *History Preferential Attachment*

1) HRF and HRF-Based Task Allocation: According to the resource caching mechanism described in Section III-A, the more frequently an agent accesses a resource in its history, the nearer the resource replica is cached to that agent; thus, the agent may have higher access to that resource. Therefore, an agent's history of accessing a resource can influence the agent's access to the resource in the future, which can be described as "history deciding the future."

When an agent accesses a resource to implement a task, a replica of the resource is cached near the agent, according to Section III-A1. However, according to the eclipse mechanism of resource caching described in Section III-A2, a resource replica may be eclipsed if it has not been used for long time. The more recently an agent accesses a resource, the more probably the resource replica remains near the agent.

We first set a start time and then compute an agent's history resource accessing factor (*HRF*) from a start time until the current time. We can measure the resource accessing history of an agent as follows.

Definition 2: Let two time points be t_0 and t_*, $t_0 < t_*$. From time t_0, the HRF of agent a_i for resource r_k at time t_* is

$$h_i^{t_0 \to t_*}(k) = \sum_{t_0 \le t \le t_*} \left(\frac{1/(t_* - t)}{\sum_{t_0 \le t \le t_*} (1/(t_* - t))} n_i^t(k) \right) \tag{6}$$

where $n_i^t(k)$ is the number of r_k's that are accessed by a_i at time t and t_0 is a predefined starting time for consideration.

Lemma 1: Given three time points, t_{00}, t_{01}, and t_*, $t_{00} < t_{01} < t_*$, we have $h_i^{t_{00} \to t_*}(k) \ge h_i^{t_{01} \to t_*}(k)$ for any agents and resources in the network.

Proof: According to Definition 2, $h_i^{t_{00} \to t_*}(k) = h_i^{t_{00} \to t_{01}}(k) + h_i^{t_{01} \to t_*}(k)$, and $h_i^{t_{00} \to t_{01}}(k) \ge 0$. Thus, we have $h_i^{t_{00} \to t_*}(k) \ge h_i^{t_{01} \to t_*}(k)$. □

From Lemma 1, now, we have a problem: If the predefined starting time is set as early as possible, then the corresponding *HRF* can be heightened. However, according to the eclipse mechanism of resource caching described in Section III-A2, if a cached resource has not been used for long time, it may be moved back toward its original inhabitation locality and even be terminated. Therefore, we should modify Definition 2; if an agent accessed a resource *frequently* and *recently*, its *HRF* for the resource is higher. We now present the definition of *standardized HRF* as follows:

$$\begin{aligned} h_i(k) &= h_i^{t_0 \to t_*}(k)/(t_* - t_0) \\ &= \left(\sum_{t_0 \le t \le t_*} \left(\frac{1/(t_* - t)}{\sum_{t_0 \le t \le t_*} (1/(t_* - t))} n_i^t(k) \right) \right) /(t_* - t_0). \end{aligned} \tag{7}$$

Obviously, the higher $h_i(k)$ is, then the more *frequently* and *recently* agent a_i accessed resource r_k.

Definition 3: Accessibility of an agent to a resource. *Given a resource r_k in the network, the set of cached replicas and original one of r_k is R_k. Now, we simply define the accessibility of agent a_i to resource r_k as follows*:

$$A_i(k) = 1/ \left(\min_{r_x \in R_k} l(P_{ix}) \right) \tag{8}$$

where P_{ix} is the shortest path from a_i to r_x and $l(P_{ix})$ is the length of path P_{ix}.

Theorem 2: Given two agents in the network, a_i and a_j, a_i and a_j accessed resource r_k to execute tasks from time t_1 to t_2. The accessibility of agent a_y to resource r_k at time t_x is denoted as $A_y^x(k)$; the difference between the accessibilities of a_i and a_j to resource r_k at time t_x is denoted as σ_x, $\sigma_x = |A_i^x(k) - A_j^x(k)|$. Now, we consider the following three situations.

1) $A_i^1(k) = A_j^1(k)$: If $h_i^{t_1\to t_2}(k) > h_j^{t_1\to t_2}(k)$, we have $A_i^2(k) \geq A_j^2(k)$.
2) $A_i^1(k) > A_j^1(k)$: If $h_i^{t_1\to t_2}(k) > h_j^{t_1\to t_2}(k)$, we have $\sigma_2 \geq \sigma_1$.
3) $A_i^1(k) < A_j^1(k)$: If $h_i^{t_1\to t_2}(k) > h_j^{t_1\to t_2}(k)$, we have $\sigma_2 \leq \sigma_1$ while $A_i^2(k) \leq A_j^2(k)$ or $A_i^2(k) > A_j^2(k)$.

Proof:

1) If $A_i^1(k) = A_j^1(k)$, at time t_1, the minimum length of the shortest path from a_i to any resource in R_k (denoted as r_{ik}) is the same as the one from a_j to any resource in R_k (which is denoted as r_{jk}). According to the resource caching mechanism, the localities of r_{ik} and r_{jk} at time t_2 are $L_{L_{rik}\to L_i}^{n_i}$ and $L_{L_{rik}\to L_i}^{n_j}$. Because $h_i^{t_1\to t_2}(k) > h_j^{t_1\to t_2}(k)$, the migration steps of r_{ik} to a_i (n_i) are greater than those of r_{jk} to a_j (n_j); thus, the length between a_i and $L_{L_{rik}\to L_i}^{n_i}$ is shorter than that between a_j and $L_{L_{rik}\to L_i}^{n_j}$. Therefore, we have $A_i^2(k) \geq A_j^2(k)$.

   The proofs of 2) and 3) are similar to 1), so here, we skip them to save space. □

In Theorem 2, we can see that higher *HRF* can definitely improve an agent's access to a resource. Therefore, the task can be allocated to the agent with the highest *HRF* for the required resource, which can reduce the communication time in (5). *The HRF-based task allocation is simply described as follows: 1) Select the principal agent with the highest HRF for required resources; 2) the principal agent searches all required resources according to Algorithm 2, selecting assistant agents; and 3) the principal agent and assistant agents cooperate to execute the task.*

2) C-HRF and C-HRF-Based Task Allocation: Each agent situates within a context in the multiagent system [15], [37], [38]; an agent's resource access can be influenced by its contextual agents' resource access histories. For an agent a_i, if one agent in a_i's context, a_j, accessed a resource, the resource will be cached toward a_j; now, a_i's access to the resource will also be influenced because a_j is in the context of a_i. The nearer a_j is to a_i, the more probably a_j's resource access history influences a_i's resource access.

Definition 4: Let two time points be t_0 and t_*, $t_0 < t_*$. From time t_0, the contextual HRF (C-HRF) of agent a_i for resource r_k at time t_* is

$$Ch_i^{t_0\to t_*}(k) = \sum_{a_j\in C_i}\left(\frac{1/d_{ij}}{\sum_{a_j\in C_i}(1/d_{ij})}h_j^{t_0\to t_*}(k)\right) \tag{9}$$

where C_i is the context of agent a_i, which can be set from its neighboring area to the whole network and d_{ij} is the distance between agent a_i and a_j in the network; $a_i \in C_i$, d_{ii} is less than d_{ij} if $a_j \neq a_i$. Similarly to (7), *we can also propose a definition of standardized C-HRF as follows*:

$$\begin{aligned} Ch_i(k) &= Ch_i^{t_0\to t_*}(k)/(t_* - t_0)\\ &= \left(\sum_{a_j\in C_i}\left(\frac{1/d_{ij}}{\sum_{a_j\in C_i}(1/d_{ij})}h_j^{t_0\to t_*}(k)\right)\right)/(t_* - t_0). \end{aligned} \tag{10}$$

From Definition 4, if an agent's contextual agents accessed a resource *frequently* and *recently*, the agent may have higher access to the resource even if it seldom accessed the resource by itself. Therefore, the task can be allocated to the agent with the highest *C-HRF* for the required resources, which is called *task allocation based on C-HRF*.

B. Present Preferential Attachment

If there are tasks queuing for an agent, the execution of the tasks will access some resources; according to the resource caching mechanism, the accessed resources will be cached toward that agent so that the agent's future access to those resources will be improved. Therefore, we can say that an agent's present access to a resource may influence the agent's access to the resource in the future, a situation in which the "present is deciding the future."

1) PRF and PRF-Based Task Allocation: Let the present team of tasks queuing for agent a_i and needing resource r_k be Q_{ik} and the size of Q_{ik} be s_{ik}. Then, we can define the present resource accessing factor (PRF) as follows.

Definition 5: The PRF of agent a_i for resource r_k is

$$p_i(k) = f(s_{ik}) \tag{11}$$

where f is a monotonically increasing function.

Theorem 3: Let two agents in the network be a_i and a_j. The present team of tasks queuing for agent a_i and needing resource r_k is Q_{ik}, and the size of Q_{ik} is s_{ik}; the present team of tasks queuing for agent a_j and needing resource r_k is Q_{jk}, and the size of Q_{jk} is s_{jk}. t_1 denotes the current time. t_2 denotes the time when the queued tasks will be finished. The accessibility of agent a_y to resource r_k at time t_x is denoted as $A_y^x(k)$; the difference between the accessibilities of a_i and a_j to resource r_k at time t_x is denoted as σ_x, $\sigma_x = |A_i^x(k) - A_j^x(k)|$. Now, we consider the following three situations.

1) $A_i^1(k) = A_j^1(k)$: If $p_i(k) > p_j(k)$, we have $A_i^2(k) \geq A_j^2(k)$.
2) $A_i^1(k) > A_j^1(k)$: If $p_i(k) > p_j(k)$, we have $\sigma_2 \geq \sigma_1$.
3) $A_i^1(k) < A_j^1(k)$: If $p_i(k) > p_j(k)$, we have $\sigma_2 \leq \sigma_1$ while $A_i^2(k) \leq A_j^2(k)$ or $A_i^2(k) > A_j^2(k)$.

Proof: The proof is similar to that of Theorem 2, so here, we skip it to save space. □

Based on Theorem 3, an agent will have higher access to a resource if it has a higher *PRF* for that resource. Therefore, the task can be allocated to the agent with the highest *PRF* for the required resources, which can reduce the communication time in (5). After the principal agent is determined based on *PRF*, the remaining process is similar to the *HRF*-based task allocation.

2) C-PRF and C-PRF-Based Task Allocation: As in Section V-A2, for an agent a_i, the resource access situation of its contextual agents' queuing tasks will influence a_i's future access to resources. The nearer a contextual agent is to a_i, the more the contextual agent's resource access situation of the tasks in its queue influences a_i's future access to resources. We present the definition of the contextual PRF (C-PRF) as follows.

Definition 6: The C-PRF of agent a_i for resource r_k is

$$Cp_i(k) = \sum_{a_j \in C_i} \left(\frac{1/d_{ij}}{\sum_{a_j \in C_i}(1/d_{ij})} p_i(k) \right) \tag{12}$$

where C_i is the context of agent a_i, which can be set from its neighboring area to the whole network and d_{ij} is the distance between agent a_i and a_j in the network; $a_i \in C_i$, d_{ii} is less than d_{ij} if $a_j \neq a_i$.

From Definition 6, if an agent's contextual agents' queuing tasks are rich, the agent may have higher future access to such resources even if the agent itself has fewer queuing tasks. Thus, a task can be allocated to the agent that has the highest C-PRF for the required resources, which is called *task allocation based on C-PRF*.

3) On Load Balancing: As mentioned earlier, if an agent possesses more queuing tasks, it may accordingly be assigned more new tasks. However, if too many tasks are crowded on to certain agents, the waiting time may outperform the benefit (reduction in communication time) brought by the resource caching of executing queuing tasks. Thus, *we should apply load balancing in task allocation when the benefit brought by preferential attachment is less than the loss brought by the waiting time of queuing tasks*. The next case study can demonstrate this situation.

Case Study 1: Let a multiagent network be $G = \langle A, E\rangle$ and the communication time between any neighbor agents be t_ε. The resource caching eclipse time period is t_*; now, there is a resource r_k. $\exists a_i \in A$, $s_{ik} = n$; $\forall\, t \in Q_{ik}$, the execution time of task t is t_t. twill access r_k for once. Now, if a task t_{new} is allocated to a_i, then t_{new} will be executed after the tasks in Q_{ik} are all finished. Then, now, the influence of Q_{ik} on t_{new} includes the following.

1) ***The benefit brought by resource caching: Because $s_{ik} = n$, resource r_k will be cached and migrated toward a_i with n steps; thus; the saved communication time by allocating t_{new} to a_i is nt_ε.***
2) ***The loss brought by resource eclipse within the waiting time: t_{new} will wait for nt_t time to execute, in which the cached resource will migrate back to its original locality for nt_t/t_* steps; thus; the wasted time is $(nt_t/t_*)t_\varepsilon$.***
3) ***The loss brought by waiting time: nt_t.***
4) Therefore, while task t_{new} is allocated to agent a_i for resource r_k and now $s_{ik} = n$, the net benefit is

$$(n - n \cdot t_t/t_*) \cdot t_\varepsilon - n \cdot t_t. \tag{13}$$

When we allocate tasks, we should perform load balancing if the value of (13) is negative. Because t_ε is usually less than t_t in reality, the load balancing should be taken into account when there is a long queue of tasks.

According to Case Study 1, we should perform load balancing when the queue of tasks is too long. Therefore, we can modify (11) as follows:

$$p_i^*(k) = \alpha(s_{ik}) \cdot p_i(k) = \alpha(s_{ik}) \cdot f(s_{ik}) \tag{14}$$

where $\alpha(s_{ik})$ is an attenuation function, $0 \leq \alpha(s_{ik}) \leq 1$; the value of $\alpha(s_{ik})$ decreases monotonically from 1 to 0 with the increase of s_{ik}.

Therefore, to perform load balancing, the definition of *C-PRF* in (12) should also be modified, shown as follows:

$$\begin{aligned} Cp_i^*(k) &= \sum_{a_j \in C_i} \left(\frac{1/d_{ij}}{\sum_{a_j \in C_i}(1/d_{ij})} p_j^*(k) \right) \\ &= \sum_{a_j \in C_i} \left(\frac{1/d_{ij}}{\sum_{a_j \in C_i}(1/d_{ij})} \left(\alpha(s_{jk}) \cdot f(s_{jk})\right) \right). \end{aligned} \tag{15}$$

Obviously, preferential attachment-based task allocation with consideration of load balancing can reduce both the waiting time and communication time in (5).

C. History–Present Combined Preferential Attachment

Now we combine the two preferential attachments, referred to as *history–present combined preferential attachment in task allocation.*

Definition 7: History–present combined resource accessing factor (HP-RF) of agent a_i for resource r_k can be defined as

$$hp_i(k) = \lambda_1 \cdot h_i(k) + \lambda_2 \cdot p_i(k). \tag{16}$$

Contextual HP-RF (C-HP-RF) of agent a_i for resource r_k can be defined as

$$C - hp_i(k) = \lambda_1 \cdot Ch_i(k) + \lambda_2 \cdot Cp_i(k). \tag{17}$$

If load balancing is applied, (16) *and* (17) *can be modified as follows*:

$$hp_i^*(k) = \lambda_1 \cdot h_i(k) + \lambda_2 \cdot p_i^*(k) \tag{18}$$

$$C - hp_i^*(k) = \lambda_1 \cdot Ch_i(k) + \lambda_2 \cdot Cp_i^*(k) \tag{19}$$

where λ_1 and λ_2 are used to determine the relative importance of the two types of preferential attachments, $\lambda_1 + \lambda_2 = 1$.

Then, the task can be allocated to the agent that has the highest *HP-RF* or *C-HP-RF* for the required resources, which is called *task allocation based on history–present combined preferential attachment*.

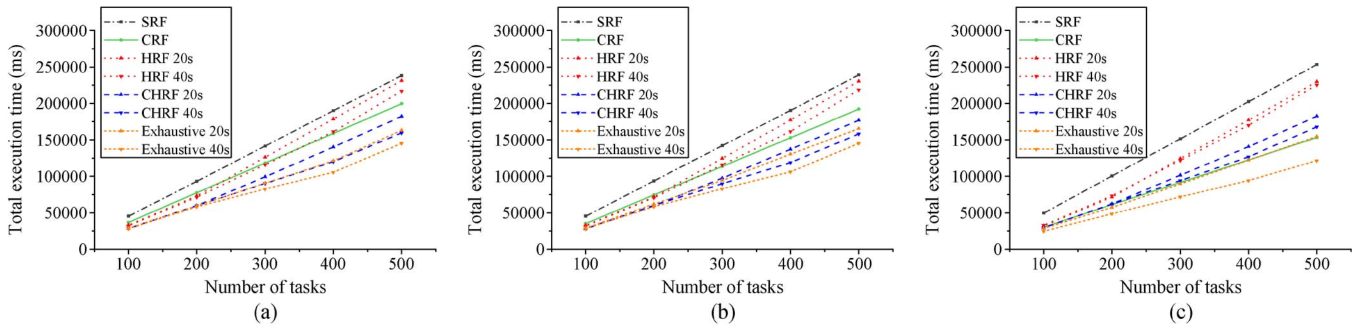


Fig. 1. Execution time comparison between SRF, CRF, exhaustive-trial, *HRF*, and CHRF models. (a) FRFS. (b) MIFS. (c) AAS.

VI. Experiment Validations and Analyses

A. Introduction of Experiments

We make experiments based on our developed simulation platform for NMASs in large-scale dynamic topology environments (*ComDysc-v1*). The platform is developed in Java (JDK 1.6) using Eclipse IDE [39] and is supported by the National High Technology Research and Development Program of China. The tasks in experiments are similar to our previous work [15]; to save paper space, we do not describe them in detail here.

When we select a principal agent in task allocation, the key is to satisfy the requirements of resources for executing tasks. Without loss of generality, we use three criteria to satisfy the different resource requirements of tasks in our experiments: 1) First required resource-first satisfy (FRFS); 2) most important resource-first satisfy (MIFS); and 3) all resources-averagely satisfy (AAS).

Task execution in multiagent systems can be described by agents' operations when accessing required resources. Therefore, we mainly compare our methods with the previous resource-based task allocation methods: 1) self-owned resource-based task allocation model in related work (SRF Model); 2) contextual resource-based task allocation model (CRF Model); and 3) the idealized approach: exhaustive-trial method.

B. Validation for History Preferential Attachment

We will now validate the history preferential attachment in task allocation by comparing the performances of the following models: SRF, CRF, exhaustive-trial, *HRF*, and CHRF. We make a series of experiments in which the number of agents is 200, each agent owns four types of resources by itself, and each task needs 20 resources. The results are shown in Fig. 1, where the x-axis denotes the number of tasks and the y-axis denotes the total execution time of tasks. Moreover, we set different withdrawal time periods of resource caching for the exhaustive-trial, *HRF*, and CHRF models; for example, *HRF* 20s denotes the *HRF* model whose withdrawal time period of resource caching is 20 s.

From the experiment results, we conclude the following: 1) *HRF* and CHRF models outperform the SRF model, which shows that the history preferential attachment in task allocation can effectively reduce communication time for accessing resources; 2) CHRF outperforms CRF in FRFS and MIFS, which shows the history preferential attachment is obviously feasible while certain resources are preferential in task execution; CRF performs very well when AAS is used because CHRF implements task allocation essentially relying on the unbalanced effects of preferential attachment, but now, AAS averages the overall required resources so that the unbalanced effects in task allocation are not obvious; 3) CHRF outperforms *HRF*, which shows that it is more advanced when the context is considered in the network; and 4) with an increase in the resource caching withdrawal time period or number of tasks, the effect of historical preferential attachment becomes more obvious.

In conclusion, the experiment results prove that history preferential attachment in task allocation can effectively improve system performance, particularly when the network context situation is considered or the number of tasks is high.

C. Validation for Present Preferential Attachment

Now, we validate the present preferential attachment in task allocation by comparing the following models: SRF, CRF, exhaustive-trial, PRF, and CPRF. The results are shown in Fig. 2, where SRF_LB, PRF_LB, and CPRF_LB apply load balancing.

From the experiment results, we conclude the following: 1) PRF and CPRF models absolutely outperform the SRF model and are close to the CRF model, which shows that present preferential attachment in task allocation can reduce communication time for resource access more than the SRF model; 2) CPRF outperforms CRF in FRFS and MIFS, which shows the present preferential attachment is obviously feasible when certain resources are preferential in task execution; CRF performs very well when AAS is used because CPRF implements task allocation essentially relying on the unbalanced effects of preferential attachment, but now, AAS averages the overall required resources so that the unbalanced effects in task allocation are relaxed; 3) CPRF outperforms PRF, which shows that it is more advanced when network context is considered; 4) with an increase in the resource caching withdrawal time period or the number of tasks, the present preferential attachment effect becomes more obvious; and 5) PRF_LB outperforms PRF, and CPRF_LB outperforms CPRF. Therefore, it is better to combine present preferential attachment and load balancing.

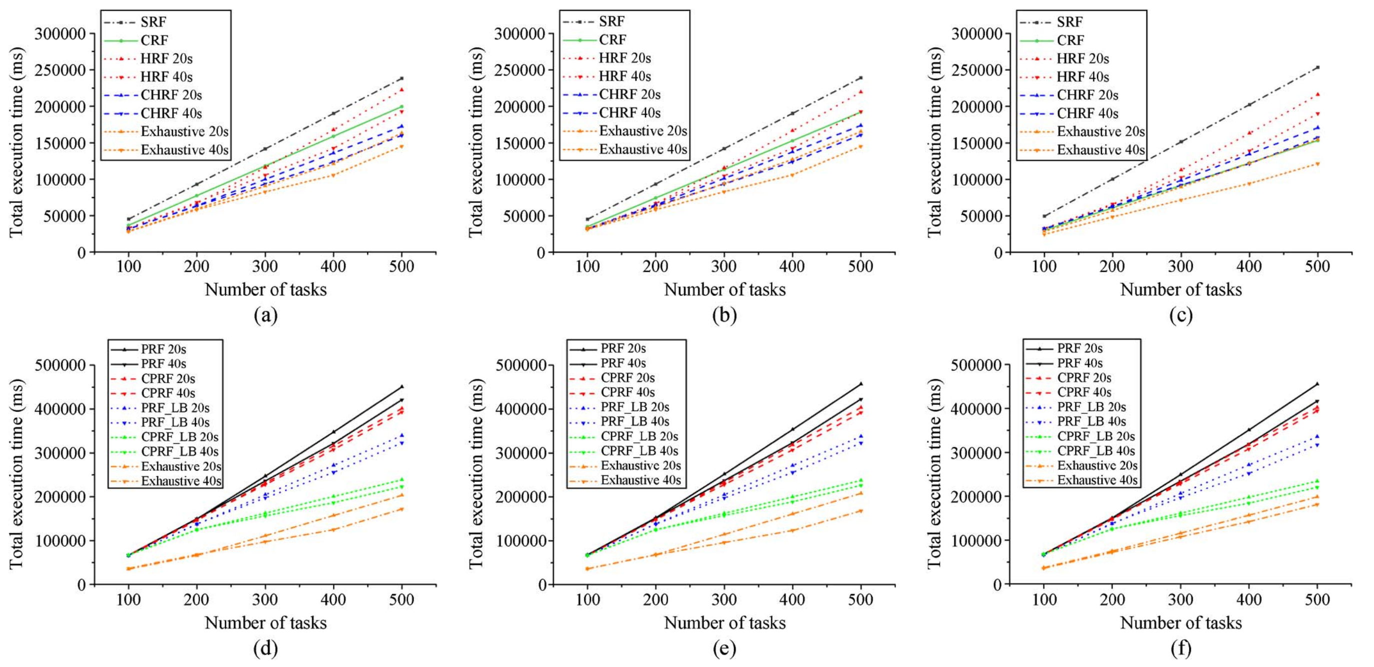


Fig. 2. Execution time comparison between SRF, CRF, exhaustive-trial, PRF, and CPRF models. (a)–(c) do not apply load balancing. (d)–(f) apply load balancing. (a) FRFS. (b) MIFS. (c) AAS. (d) FRFS. (e) MIFS. (f) AAS.

In conclusion, the experiment results prove that the present preferential attachment in task allocation can effectively improve system performance, particularly when the network context situation is considered or the number of tasks is high. Moreover, it is better to compromise between preferential attachment and load balancing while there are too many waiting tasks. Therefore, preferential attachment and load balancing can sometimes be compatible in task allocation.

D. Validation for History–Present Combined Preferential Attachment

Now, we validate the history–present combined preferential attachment in task allocation by comparing the performances of the following models: SRF, CRF, exhaustive-trial, HPRF, and CHPRF. The results are shown in Fig. 3, where HPRF_LB and CHPRF-LB are the HPRF and CHPRF models applying load balancing.

From the experiment results, we conclude the following: 1) HPRF and CHPRF models absolutely outperform the SRF model and are close to the CRF model, which shows that the history–present combined preferential attachment of task allocation can reduce communication time for resource access more than the SRF model; 2) CHPRF outperforms CRF in FRFS and MIFS, which shows the history–present combined preferential attachment is obviously feasible while certain resources are preferential in the task execution; CRF performs very well when AAS is used because CHPRF implements task allocation essentially relying on the unbalanced effects of preferential attachment, but now, AAS averages the overall required resources so that the unbalanced effects in task allocation are relaxed; 3) CHPRF outperforms HPRF, which shows that it is more advanced when the network context is considered; 4) with an increase in the resource caching withdrawal time period or number of tasks, the history–present combined preferential attachment effect becomes more obvious; 5) the HPRF model outperforms both PRF and *HRF* models, and therefore, it is better if we integrate the history and present preferential attachments; and 6) HPRF_LB outperforms HPRF, and CHPRF_LB outperforms CHPRF; therefore, it is better if we can make a compromise between the history–present combined attachment and load balancing while there are many tasks waiting to execute. Moreover, the performance gap between HPRF_LB and HPRF (or between CHPRF_LB and CHPRF) is smaller than that between PRF_LB and PRF (or between CPRF_LB and CPRF in Fig. 2) because HPRF includes the history preferential attachment which does not consider load balancing.

In conclusion, the experiment results prove that the history–present combined preferential attachment in task allocation can effectively improve system performance, particularly when the network context situation is considered and the number of tasks is high. Moreover, the integration of history and present preferential attachments outperforms either history or present preferential attachment alone in task allocation.

E. Comparison With the Idealized Approach

We will compare our approaches with the idealized approach (exhaustive-trial method) by summarizing the related results of the experiments in Section VI-B–D, shown in Table I. The comparison in Table I is measured as follows. Let the task execution time using our approach be x and the time using the idealized approach be y. Their comparison is $c = 1 - ((x - y)/y)$; we can compute the mean of such values in a series of experiments. From Table I, we can see the following: 1) When load balancing is not applied, the mean utilities produced by our approaches are within almost 70%–95% of those produced by

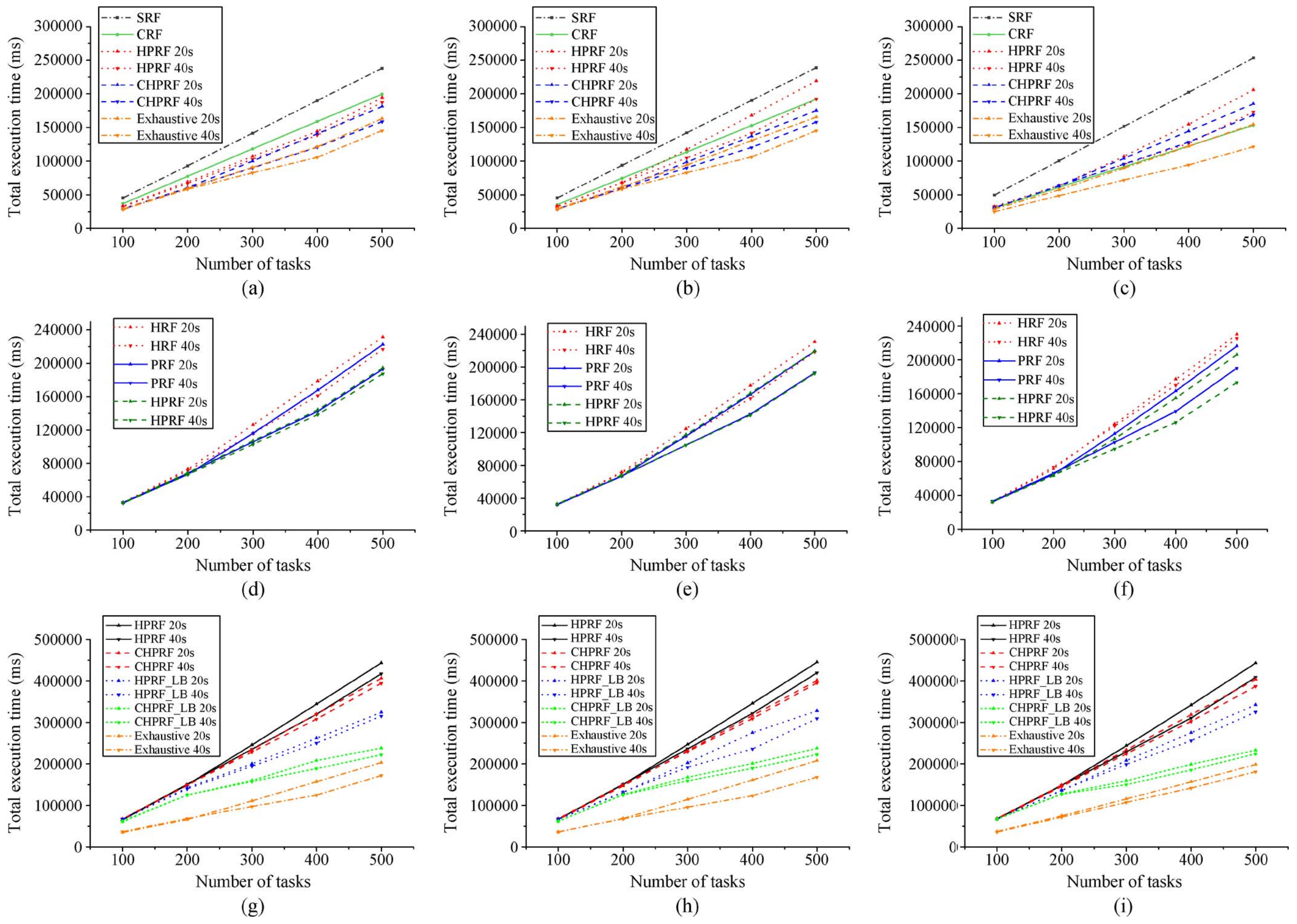


Fig. 3. Execution time comparison between SRF, CRF, exhaustive-trial, PRF, *HRF*, HPRF, and CHPRF models. (a) FRFS. (b) MIFS. (c) AAS. (d) FRFS. (e) MIFS. (f) AAS. (g) FRFS. (h) MIFS. (i) AAS.

TABLE I
COMPARISON BETWEEN OUR APPROACHES AND THE IDEALIZED APPROACH (EXHAUSTIVE-TRIAL METHOD)

Resource satisfaction	Experimental stages	HRF	PRF	HPRF	PRF-LB	HPRF-LB
FRFS	*Mean of late stages*	*0.891718*	*0.920678*	*0.898327*	*0.764232*	*0.767864*
	Mean of all stages	0.927152	0.893712	0.924346	0.422324	0.446571
MIFS	*Mean of late stages*	*0.920648*	*0.921125*	*0.925784*	*0.756029*	*0.76687*
	Mean of all stages	0.959387	0.925576	0.953564	0.432339	0.450073
AAS	*Mean of late stages*	*0.719294*	*0.800006*	*0.706501*	*0.802319*	*0.793195*
	Mean of all stages	0.792458	0.800264	0.767935	0.505692	0.506209

the idealized approach, which denotes that our approaches are effective; 2) when load balancing is considered, our approaches cannot produce good results in the early stages of experiments; the potential reason for this is that the highly burdened agents do not have absolute dominance over resource access, so the benefit brought by preferential attachment is less than the loss caused by the waiting time of queuing tasks; and 3) when load balancing is applied, our approaches produce mean utilities within 76%–80% of those produced by the idealized approach at the late stages of experiments. The potential reason for this is that, now, the highly burdened agents have absolute dominance over resource access, so the benefit brought by preferential attachment is more than the loss caused by the waiting time of queuing tasks.

VII. CONCLUSION AND DISCUSSION

In practical NMASs such as grids and P2P systems, communication time for accessing required resources is crucial to system performance. In this paper, we are inspired by the idea that "the rich get richer" to investigate task allocation in the NMASs with resource caching. We find that task allocation based on preferential attachment can effectively reduce agents' communication time to access resources for executing tasks. On the other hand, we also find that it is better to make a compromise between preferential attachment and load balancing when there are too many tasks waiting for execution. Therefore, the ideas of "rich get richer" and "winner does not take all" may sometimes be compatible in the task allocation of NMASs with resource caching. Furthermore, our model is implemented by simply relying on the agents' experiences of executing tasks and does not need to know the accurate resource distribution information in the system. Thus, our model could be well applied in large-scale dynamic situations in which accurate resource information is difficult to acquire timely.

Regarding the generality and future work of our model, the following are several aspects for discussion.

1) For simplicity, this paper abstracts the distance between two agents as the hops between them and assumes the distances between any two adjacent agents are the same. Such assumption is reasonable in some applications when the hops between two agents are crucial to their communication, such as sensor networks. However, in other situations, distances in terms of resources may not be the same between adjacent agents; they could be a complex function that depends on the type of tasks to be performed. In such a situation, we simply need to modify the concept of distance and revise the resource-searching algorithm by taking the meaning of distance into account. Therefore, our model can be extended into other real situations in which the distances have more complex meanings.
2) In our model, the agents are assumed to differ essentially in their access to resources. Such an assumption is reasonable in some real situations in which all agents are identical. However, in some circumstances, agents may have different computable functions or other resources that cannot be cached, such as CPU power and memory storage. In such a situation, when we perform task allocation, we can select the principal agent that can satisfy the required computable functions and has the highest *HRF* (or PRF) for required resources. Therefore, our presented model can be extended into other situations in which agents have different computable functions.
3) This paper is only concerned with cooperative agents, i.e., all agents can contribute their idle resources and corporately work together to accomplish tasks. However, in reality there are some selfish multiagent systems in which each agent optimizes its own object independently of the others [13], [21]. In the task allocation of such selfish multiagent systems, automated negotiation [13] or noncooperative game [21] is used in related benchmark works, and equilibrium can be obtained by a distributed noncooperative policy. Therefore, in future work, to extend our model into situations in which agents are selfish, we will also introduce the automated negotiation or game theory in the resource search process when the agents can afford the additional computing and communication costs.
4) Our model is designed for NMASs in which resource access can be improved through caching by executing tasks and communication time for accessing resources is crucial to system performance. In fact, the "rich get richer" method can also be generalized to other systems where preferential attachments exist, e.g., agents have self-learning abilities and can improve their capacities through executing tasks. Therefore, our future work will try to improve the generalizability of the "rich get richer" idea in other NMASs.

References

[1] J. Liu, X. Jin, and Y. Wang, "Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 7, pp. 586–598, Jul. 2005.

[2] L. Yin and G. Cao, "Supporting cooperative caching in ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 5, no. 1, pp. 77–89, Jan. 2006.

[3] Y. Cardenas, J.-M. Pierson, and L. Brunie, "Uniform distributed cache service for grid computing," in *Proc. 16th Int. Workshop DEXA*, Copenhagen, Denmark, Aug. 22–26, 2005, pp. 351–355.

[4] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt, "Improving data access in P2P systems," *IEEE Internet Comput.*, vol. 6, no. 1, pp. 58–67, Jan./Feb. 2002.

[5] H. K. Pillai and S. Shankar, "A behavioral approach to control of distributed systems," *SIAM J. Control Optim.*, vol. 37, no. 2, pp. 388–408, Mar. 1999.

[6] B. Bulka, M. Gaston, and M. des Jardins, "Local strategy learning in networked multi-agent team formation," *J. Autonomous Agents Multi-Agent Syst.*, vol. 15, no. 1, pp. 29–45, Aug. 2007.

[7] S. Abdallah and V. Lesser, "Multiagent reinforcement learning and self-organization in a network of agents," in *Proc. 6th Int. Conf. AAMAS*, Honolulu, HI, May 14–18, 2007, pp. 172–179.

[8] W. Rao, L. Chen, A. W.-C. Fu, and G. Wang, "Optimal resource placement in structured peer-to-peer networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 7, pp. 1011–1026, Jul. 2010.

[9] J. Zhao, P. Zhang, G. Cao, and C. R. Das, "Cooperative caching in wireless P2P networks: Design, implementation, and evaluation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 2, pp. 229–241, Feb. 2010.

[10] Y. Jiang, J. Hu, and D. Lin, "Decision making of networked multiagent systems for interaction structures," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 6, pp. 1107–1121, Nov. 2011.

[11] Y. Jiang and Z. Li, "Locality-sensitive task allocation and load balancing in networked multiagent systems: Talent versus centrality," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 822–836, Jun. 2011.

[12] A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive load balancing: A study in multi-agent learning," *J. Artif. Intell. Res.*, vol. 2, pp. 475–500, 1995.

[13] B. An, V. Lesser, and K. M. Sim, "Strategic agents for multi-resource negotiation," *J. Autonom. Agents Multi-Agent Syst.*, vol. 23, no. 1, pp. 114–153, Jul. 2011.

[14] B. An, F. Douglis, and F. Ye, "Heuristics for negotiation schedules in multi-plan optimization," in *Proc. 7th Int. Conf. AAMAS*, Estoril, Portugal, May 12–16, 2008, pp. 551–558.

[15] Y. Jiang and J. Jiang, "Contextual resource negotiation-based task allocation and load balancing in complex software systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 641–653, May 2009.

[16] S. Kraus and T. Plotkin, "Algorithms of distributed task allocation for cooperative agents," *Theoretical Comput. Sci.*, vol. 242, no. 1/2, pp. 1–27, Jul. 2000.

[17] S. Kraus, O. Shehory, and G. Taase, "Coalition formation with uncertain heterogeneous information," in *Proc. 2nd Int. Conf. AAMAS*, Melbourne, Australia, Jul. 14–18, 2003, pp. 1–8.

[18] S. D. Ramchurn, C. Mezzetti, A. Giovannucci, J. A. Rodriguez-Aguilar, R. K. Dash, and N. R. Jennings, "Trust-based mechanisms for robust and efficient task allocation in the presence of execution uncertainty," *J. Artif. Intell. Res.*, vol. 35, no. 1, pp. 119–159, May 2009.

[19] M. J. Mataric, G. S. Sukhatme, and E. H. Qstergaard, "Multi-robot task allocation in uncertain environments," *Autonom. Robots*, vol. 14, no. 2/3, pp. 255–263, Mar.–May 2003.

[20] K. Lerman, C. Jones, A. Galstyan, and M. J. Mataric, "Analysis of dynamic task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 25, no. 3, pp. 225–241, Mar. 2006.

[21] D. Grosu and A. T. Chronopoulos, "Noncooperative load balancing in distributed systems," *J. Parallel Distrib. Comput.*, vol. 65, no. 9, pp. 1022–1034, Sep. 2005.

[22] J. Jiang and X. Xia, "Prominence convergence in the collective synchronization of situated multi-agents," *Inform. Process. Lett.*, vol. 109, no. 5, pp. 278–285, Feb. 2009.

[23] M. E. J. Newman, "Clustering and preferential attachment in growing networks," *Phys. Rev. E*, vol. 64, p. 025 102(R), 2001.

[24] B. Hong and V. K. Prasanna, "Adaptive allocation of independent tasks to maximize throughput," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 10, pp. 1420–1435, Oct. 2007.

[25] Y.-C. Jiang and J. C. Jiang, "A multi-agent coordination model for the variation of underlying network topology," *Expert Syst. Appl.*, vol. 29, no. 2, pp. 372–382, Aug. 2005.

[26] M. Erdmann and T. Lozano-Perez, "On multiple moving robots," *Algorithmica*, vol. 2, no. 4, pp. 477–521, 1987.

[27] D. Palmer, M. Kirschenbaum, J. Murton, and K. Zajac, "Decentralized cooperative auction for multiple agent task allocation using synchronized random number generators," in *Proc. IEEE/RSJ, Int. Conf. Intell. Robots Syst.*, Las Vegas, NV, Oct. 2003, pp. 1963–1968.

[28] S. Fjuita and V. R. Lesser, "Centralized task distribution in the presence of uncertainty and time deadlines," in *Proc. 2nd ICMAS*, Kyoto, Japan, Dec. 10, 1996, pp. 87–94.

[29] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.*, vol. 101, no. 1/2, pp. 165–200, May 1998.

[30] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1104–1113, Dec. 1980.

[31] S. Aknine, S. Pinson, and M. F. Shakun, "An extended multi-agent negotiation protocol," *J. Autonom. Agents Multi-Agent Syst.*, vol. 8, no. 1, pp. 5–45, Jan. 2004.

[32] K.-P. Chow and Y.-K. Kwok, "On load balancing for distributed multiagent computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 8, pp. 787–801, Aug. 2002.

[33] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. Yang, and D. A. Bader, "Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 485–497, Apr. 2007.

[34] D. M. Pennock, G. W. Flake, S. Lawrence, E. J. Glover, and C. L. Giles, "Winners don't take all: Characterizing the competition for links on the web," in *Proc. Nat. Acad. Sci.*, Apr. 2002, vol. 99, no. 8, pp. 5207–5211.

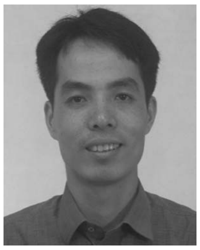
[35] K. Ranganathan and I. Foster, "Design and evaluation of dynamic replication strategies for a high-performance data grid," in *Proc. Int. Conf. Comput. High Energy Nucl. Phys.*, Beijing, China, Sep. 3–7, 2001, pp. 712–715.

[36] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Chemical reaction optimization for task scheduling in grid computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1624–1631, Oct. 2011.

[37] A. Padovitz, S. W. Loke, and A. Zaslavsky, "Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 4, pp. 729–742, Jul. 2008.

[38] X. Fan, M. McNeese, B. Sun, T. Hanratty, L. Allender, and J. Yen, "Human-agent collaboration for time stressed multi-context decision making," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 2, pp. 306–320, Mar. 2010.

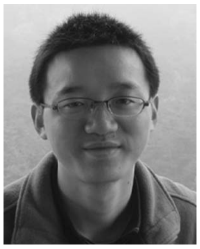
[39] Eclipse.org home, 2011. [Online]. Available: http://www.eclipse.org/



Yichuan Jiang (M'07) received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2005.

He is currently a Professor at the School of Computer Science and Engineering, Southeast University, Nanjing, China. He has published more than 70 scientific articles in refereed journals and conference proceedings, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *Journal of Parallel and Distributed Computing*, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS PART A: SYSTEMS AND HUMANS, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS PART C: APPLICATIONS & REVIEWS, *Proceedings of the International Joint Conferences on Artificial Intelligence*,and *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. His main research interests include multiagent systems, social networks, and complex distributed systems.

Dr. Jiang is a Senior Member of China Computer Federation and Chinese Institute of Electronics, a member of the editorial board of Advances in Internet of Things, an Editor of the International Journal of Networked Computing and Advanced Information Management, an Editor of Operations Research and Fuzziology, and a member of the editorial board of the Chinese Journal of Computers.



Zhichuan Huang received the B.E. degree in information engineering from Southeast University, Nanjing, China, in 2009, where he is currently working toward the M.E. degree.

His main research interests include social networks and multiagent systems.