2013 IEEE 25th International Conference on Tools with Artificial Intelligence

Migration Cost-Sensitive Load Balancing for Social Networked Multiagent Systems with Communities

Wanyuan Wang, Yichuan Jiang*, *IEEE Senior Member*
Laboratory for Complex Systems and Social Computing,
School of Computer Science and Engineering, Southeast University, Nanjing 211189, China.
**Corresponding author, Email: yjiang@seu.edu.cn*

***Abstract*—In the past, many approaches have been devised to address the load balancing problem for social networked multiagent systems (SN-MASs). However, few of these approaches consider the migration cost incurred when migrating tasks for load balancing; moreover, current SN-MASs often consist of communities, and the migration costs of intra-community and intercommunity transfers are heterogeneous. To minimize the load imbalance of agents and to incur the least migration cost, this paper introduces a net profit-based load balancing mechanism. In this mechanism, each load balance process (i.e., migrating a task from one agent to another agent) is associated with a net profit value which depends on the benefit it gains by making a contribution to alleviating the system load unfairness and the cost of migrating the task. The agents always perform the optimal load balance process that has the maximum net profit value, thereby improving system performance, as well as reducing the migration cost. Our simulations show that our approach not only guarantees that agents can undertake fair loads but also reduces the overhead migration costs compared with the previous load balancing approaches that ignore the cost of migrating the task.**

***Keywords*-multiagent systems; social networks; load balancing; community; migration cost; net profit**

I. Introduction

Currently, social networks play the role of the underlying interaction medium for many large-scale multiagent systems, which can be called social networked multiagent systems (SN-MASs) [1][2]. In SN-MASs, because tasks are distributed to agents in a decentralized manner and agents are heterogeneous in capacities, tasks might vary greatly in waiting time at the agents [2-7]. Migragting the excess tasks on the heavy-burdened agents to the light-burdened agents through social interactions to make each agent undertake the load proportional to its capacity can be generalized as load balancing for SN-MASs [3-5].

In the past, many approaches have been devised to address the load balancing problem in networked multiagent systems [2-7]. Existing approaches primarily focus on minimizing the load imbalance of agents, i.e., minimizing the waiting time of tasks at agents; however, they ignore the migration cost incurred when migrating tasks which might play an important role in load balancing [8][9]. For example, in a transportation system, if a transportation company transports some commodities to another company by truck to alleviate its heavy-burdened level, it will have to consider the fuel expenses consumed by trucks. Moreover, current SN-MASs often consist of communities, and the migration costs of intra-community and intercommunity transfers are heterogeneous [11-13]. For example, in the transportation systems, the volumes of fuel consumed by intra-city (community) and inter-city transportation for transporting commodities are heterogeneous because the fact that the physical distances of intra-city and inter-city trips are heterogeneous.

With this motivation, in this study, we investigate the community-aware load balancing problem in SN-MASs. We assume that agents are partitioned into communities, and there exist task migration cost between agents; moreover, the migration costs of intra-community and intercommunity transfers are heterogeneous. Under this community-aware scenario, a migration cost-sensitive load balancing approach should not only strive to ensure fair load distribution over agents but also should aim to minimize the migration costs incurred by migrating tasks. To satisfy the two objectives simultaneously, in this paper, we introduce a net profit-based load balancing mechanism. This mechanism functions by associating a net profit value with each load balance process (i.e., migrating a task from one agent to another agent). The net profit of a load balance process depends on the benefit it gains by making a contribution to alleviating the system load unfairness and the cost lost by migrating the task between agents. The agents always perform the optimal load balance process that has the maximum net profit value, thereby decreasing system load unfairness as well as reducing the system overhead migration costs.

II. Related Work

Many load balancing studies have addressed the minimization of tasks' waiting time at agents. For example, in [16], the researchers devise a bidding-based mechanism to minimize the mean job execution time. In their mechanism, the agent with a high load assigns its excess tasks to the optimal bidder agent which has the lightest load. A mobile agent-based load balancing model is proposed in [4] where a task (agent) probabilistically decides to join or leave a node according to the number of tasks waiting at that node. It proves that perfect load balancing exists if and only if agents have the complete information about the entire system. Hu and Klefstad utilize a distributed balanced tree technique to make the system converge to the global optimal load balancing state [10]. These models all ignore the heterogeneity of agents' capacities.

For computing in heterogeneous systems, Hui and Chanson propose a decentralized hydrodynamic model to make agents undertake loads proportional to their capacities [5]. On the other hand, Jiang and Jiang [6] devise a centralized load balancing approach to reduce the waiting time of tasks, where if the number of tasks queuing at an agent becomes larger, the central authority will decrease the agent's probability of being allocated new tasks in the future [2][7]. However, all these methods only aim at minimizing the load imbalance of agents, ignoring other overhead costs (e.g., the communication cost between tasks and the task migration cost between agents) during load balancing.

Other related works with the aim of reducing system communication cost have also been studied. For example, in [3], Chow and Kwok introduce a communication-based load balancing algorithm to reduce the inter-machine communication cost of agents. They propose that if an agent has a higher computation or communication workload, the agent will be likely to migrate to another machine where it has a lower inter-machine communication load. A game theory-based model is presented in [14] to provide the fair response times of all the tasks and to minimize the system's expected communication delay. These models both assume that agents have the complete information of the entire system. While in our study, agents can only exchange information with their local immediate neighbors, which is better suited to real-world applications [1][5][10].

III. PROBLEM DESCRIPTION

We formulate the community-aware load balancing problem as follows. First, we assume that there is a society of agents organized by a community-aware social network.

Definition 3.1 (Community-aware social networks). *A community-aware social network, CA-SN, is defined as $<A, C, E>$, where $A=\{a_1, ..., a_m\}$ is the set of agents. The communities $C=\{C_1, ..., C_q\}$ form a partition of these agents. $\forall(a_i, a_j)\in E$ indicates the existence of a direct social interaction between a_i and a_j, and each interaction (a_i, a_j) is associated with a weight $w(a_i, a_j)$, which represents the migration cost of one unit load between a_i and a_j. Particularly, each agent $a_i\in A$ is aware of its community attribute: which community it belongs to itself and which communities its neighbors $N_i(N_i=\{a_j|(a_i, a_j)\in E\})$ belong to*[1].

Each agent $a_i\in A$ then can be defined by a 2-tuple $<p_i, com(a_i)>$, where the capacity function p_i:$A\to\mathbb{R}$ represents agent a_i's capacity to process tasks. The community function $com(a_i)$: $A\to C$ represents the community that agent a_i belongs to.

Second, we suppose that there is a set of tasks $T=\{t_1, ..., t_n\}$ distributed over these agents. Then, each task $t_j\in T$ can be defined by a 2-tuple $<l(t_j), loc(t_j)>$, where the load function $l(t_j)$: $T\to\mathbb{R}$ indicates the process time the task t_j needs. The location function $loc(t_j)$: $T\to A$ indicates the agent where t_j is queuing. The tasks that queue at agent a_i are stored in the local task set $T(a_i)$ of agent a_i, i.e., $T(a_i)=\{t_j|loc(t_j)=a_i\}$. The total load L_i that agent a_i undertakes is then given by the total load of its local tasks, i.e., $L_i=\sum_{t_j\in T(a_i)} l(t_j)$. An agent a_i's utilization degree represents the percentage of capacity that has been used. This utilization, u_i, is given by the ratio between the total load (L_i) it undertakes and its capacity (p_i) for processing task, i.e., $u_i=L_i/p_i$. Furthermore, we assume that each agent a_i is limited to other agents' load and capacity information within its local domain Ω_i, which consists of itself and its immediate neighbors N_i.

The primary objective of traditional load balancing approaches for SN-MASs is to ensure fair load distribution over agents proportional to their capacities, that is, the utilization $u_i^{'}$ of each agent a_i at system global load fairness state should be equal to system average utilization $\bar{u}$, i.e.,

$$u_i^{'} = L_i^{'}/p_i = \bar{u} = \sum_{t_j\in T} l(t_j)/\sum_{a_k\in A} p_k \qquad (1)$$

However, under the community-aware SN-MASs scenario, when an agent a_i migrates a certain local task $t_k\in T(a_i)$ to its certain neighbor $a_j\in N_i$, it will incur the overhead migration cost $w(a_i, a_j)\cdot l(t_k)$. Therefore, the other objective of a migration cost-sensitive load balancing approach for the community-aware SN-MASs is to minimize the migration cost incurred by load balancing.

To minimize the load unfairness of agents with the least migration cost, in this paper, we propose a net profit-based load balancing mechanism where each load balance process (e.g., migrating a task t_k from one agent a_i to another agent a_j) is associated with a net profit value which depends on the benefit it gains by making a contribution to alleviating system load unfairness and the cost of migrating the task t_k from a_i to a_j. The agents always perform the optimal load balance process that has the maximum net profit value.

IV. LOAD BALANCING MODEL

We are mainly concerned with the social position of agents in a CA-SN when dealing with load balancing. The social position of agents can be categorized into the internal and boundary positions.

Definition 4.1 (Social position of agents). *Given a CA-SN=$<A, C, E>$, an agent $a_i\in A$ is denoted as the internal agent if all of its neighbors belong to the same community that includes a_i. The agent a_i is denoted as the boundary agent if its certain neighbor belongs to a different community*

[1]This assumption is plausible in a real-world scenario, in which each individual knows which community it joins itself, and which communities its close friends join [1].

than a_i's. The neighbors that belong to the same community as a_i's are called its intra-community neighbors (N_i^{intra}). Otherwise, they are called the intercommunity neighbors (N_i^{inter}) of a_i.

As the internal and boundary agents possess different social positions, they might play different roles in load balancing. For example, the boundaries are able to communicate with the agents of other communities, while it is impossible for the internals. Thus, it is necessary to devise different mechanisms for the internal and boundary agents.

A. Internal Load Balancing Mechanism

It is known that the first phase of load balancing is load evaluation [3]: each agent aggregates its local domain's load and capacity information, detects its load imbalance status and decides whether to invoke load balancing. The status of the internal agent can be defined as follows:

Definition 4.2 (Load status of internal agent). *For an internal agent a_i, the average utilization, $\bar{u}_i$, of its local domain Ω_i, is given by:*

$$\bar{u}_i = \sum_{a_j\in\Omega_i} L_j \Big/ \sum_{a_j\in\Omega_i} p_j \tag{2}$$

Agent a_i is denoted as an overloaded agent if its own utilization exceeds its domain average utilization, i.e., $u_i>\bar{u}_i$, an under-loaded agent if $u_i<\bar{u}_i$ and otherwise a neutral one.

If an internal agent a_i undertakes more tasks than it can process and becomes overloaded, the tasks queuing at a_i might experience a much longer waiting time. In this scenario, it may be desirable for a_i to transfer its excess tasks to its under-loaded neighbors. Consequently, we propose that only the overloaded agents perform load balancing; neutral and under-loaded agents do not perform any load balancing (Remark 4.1 demonstrates the other advantage of this design).

If an internal agent a_i has detected that it is in the overloaded state and attempts to perform load balancing, it needs to select a load balance strategy (i.e., which local task should be selected for migration and which target neighbor agent should receive the migrated task) to execute.

Definition 4.3 (Load balancing strategy space of internal agent). *If an internal agent a_i finds it is feasible to migrate its local task $t_k\in T(a_i)$ to its neighbor agent $a_j\in N_i$, we denote this type of migration as a load balance strategy $s_i(a_j,t_k)$ of a_i. The load balancing strategy space of a_i, S_i, then can be defined as: $S_i=\{s_i(a_j,t_k)|a_j\in N_i\wedge t_k\in T(a_i)\}$.*

Before the internal overloaded agent a_i executes a feasible load balance strategy, it needs to decide whether this load balance strategy is worth executing, i.e., how many benefits will be gained and what the cost will be?

Suppose that there is no load balancing for the internal agent a_i. The utilization standard deviation, SD_i, of Ω_i is:

$$SD_i = \big(\sum_{a_q\in\Omega_i}(u_q-\bar{u}_i)^2/|\Omega_i|\big)^{\frac{1}{2}} \tag{3}$$

Now suppose that there is load balancing under strategy $s_i(a_j,t_k)\in S_i$ executed by a_i. The utilization standard deviation of Ω_i, SD_i^*, after load balancing then becomes:

$$SD_i^* = \big(\sum_{a_q\in\Omega_i}(u_q^*-\bar{u}_i)^2/|\Omega_i|\big)^{\frac{1}{2}} \tag{4}$$

The value u_q^* represents the utilization of the domain agent $a_q\in\Omega_i$ after load balancing which is computed as follows: if $q=i$, $u_q^*=u_q-l(t_k)/p_q$; if $q=j$, $u_q^*=u_q+l(t_k)/p_q$; otherwise, $u_q^*=u_q$. SD_i^* can be computed quickly using an expression derived from Equation (4):

$$SD_i^*=\big[(SD_i)^2+\big(y(2u_j+y-2\bar{u}_i)-x(2u_i-x-2\bar{u}_i)\big)/|\Omega_i|\big]^{\frac{1}{2}} \tag{5}$$

where x=$l(t_k)/p_i$ and y=$l(t_k)/p_j$.

Recall that the main objective of a load balancing approach is to make each agent a_i undertake the load (L_i) proportional to its capacity (p_i), i.e., to minimize the utilization (u_i=L_i/p_i) imbalance of agents. Thus, we can intuitively define the benefit of a load balance process as the difference between SD_i and SD_i^*.

Definition 4.4 (Load balancing benefit of internal agent). *The benefit to the internal agent a_i under the load balance strategy $s_i(a_j,t_k)\in S_i$, $g_i(s_i)$, is defined as: $g_i(s_i)$=1–SD_i^*/SD_i, where SD_i is the utilization standard deviation of its domain Ω_i before load balancing and SD_i^* is that after the load balance process $s_i(a_j,t_k)$.*

Remark 4.1. *The overloaded agent gains a larger benefit from performing load balancing than the neutral or the under-loaded agents, and it prefers to migrate its excess load to its under-loaded neighbors.*

Recall that the other objective of the migration cost-sensitive load balancing approach is to minimize the migration cost during load balancing.

Definition 4.5 (Load balancing cost of internal agent). *The cost to the internal agent a_i under the load balance strategy $s_i(a_j,t_k)\in S_i$, $l_i(s_i)$, is simply defined as the overhead migration cost lost by migrating the task t_k from a_i to a_j, i.e., $l_i(s_i)$=$w(a_i,a_j)\cdot l(t_k)$.*

If an internal overloaded agent decides to execute a load balance process, it should not only consider the benefit gained by the process's contribution to system load fairness but also consider the cost of migrating the task. Here, we use the net profit concept (which is used in the economics as an important index for business parties' decision making [15]) to represent the significance of the load balance processes.

Definition 4.6 (Load balancing net profit of internal agent). *The net profit to the internal agent a_i under the load balance process $s_i(a_j,t_k)\in S_i$, net-pro$_i(s_i)$, is defined as its cost, $l_i(s_i)$, subtracted from its benefit $g_i(s_i)$, i.e.,*

$$net\text{–}pro_i(s_i)=\alpha g_i(s_i)-l_i(s_i) \tag{6}$$

The coefficient $\alpha(\alpha{>}0)$ determines the influence of the benefit of the internal agent's load balance process. In general, if an internal overloaded agent a_i decides to perform load balancing, it always executes the optimal strategy $s_i^*\in S_i$ that has the maximum net profit value, i.e., for any alternative strategy $s_i'\in S_i$, net-pro$_i(s_i')\leq$ net-pro$_i(s_i^*)$.

It is worth noting that if the benefit of a load balance process is negative, it makes no contribution to system load

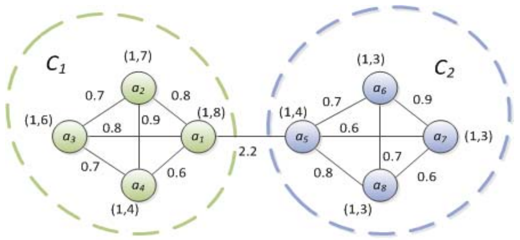

Figure 1: A toy example of community-aware social network consisting of two communities C_1 and C_2.

fairness. To avoid the unnecessary operations, we require the load balance process that to be executed to be with positive benefit value. A formal description of the internal agent' load balancing procedure can be seen in Algorithm 1.

Algorithm 1. Internal Load Balancing Algorithm (a_i,Q)

```
/* $a_i$: the internal agent where load balancing executed; */
/* $Q$: the migration cost. */
1. Set $b$=0; agent $a_i$ computes its own utilization $u_i$ and its
   domain average utilization $\bar{u}_i$ by Equation (2).
2. While ($u_i > \bar{u}_i$ &&$b \neq 1$) do
3.    Select the optimal strategy $s_i^*(a_j, t_k)$ by Equation (6).
4.    If ($g_i(s_i^*) > 0$), then
5.        Execute $s_i^*$; $Q = Q + w(a_i, a_j) \cdot l(t_k)$.
6.    Else $b$=1.
7. End While
```

B. Boundary Load Balancing Mechanism

In many real social networks with explicit communities, the migration cost of the intercommunity transfer is always larger than the migration cost of the intra-community transfer [12][13]. If agents only rely on the above internal load balancing mechanism, they will be more likely to balance loads with their intra-community neighbors, which will deteriorate the system global load fairness. We give an illustration example as follows.

Example 4.1. In Figure 1, each agent is associated with two variables. The former indicates its capacity and the latter indicates the load it undertakes. Each edge (a_i,a_j) is associated with a weight, which indicates the cost of migrating a unit load between agents a_i and a_j. According to the load balancing mechanism presented in Section IV-A, the overloaded agent a_1 would migrate 2 unit loads to agent a_4 to maximize its net profit. After executing this load balance process, all the agents in communities C_1 and C_2 then enter into a local domain equilibrium state. However, from the system global perspective, this distribution is far from fair (The utilization[2] of community C_1 is 6.25, while the utilization of C_2 is 3.25.)

Remark 4.2. *An overloaded agent gains a higher load balance benefit from apportioning its excess load to a community of under-loaded agents than unloading the excess tasks to a single under-loaded agent.*

[2]Throughout this paper, the capacity (load) of a community refers to the total capacity (load) of the agents that belong to that community, and the utilization of that community is given by the ratio between the load and capacity of that community.

Therefore, it is very much necessary to have the boundary agent migrate its community's excess load to its neighbor communities, even with the costly migration. To address this issue, first, we have each boundary agent record the average load and capacity values of the community it belongs to. Without complete community information, a boundary agent a_i may try to build estimations of the average load and capacity of its community $com(a_i)$.

Definition 4.7 (Estimations of community average load and capacity). *For each boundary agent a_i, its estimations of the average load (L_i^E) and average capacity (p_i^E) of the community it belongs to are given by the average load and capacity values of itself and its intra-community neighbors, i.e.,* $L_i^E = \frac{L_i + \sum_{a_j \in N_i^{intra}} L_j}{|N_i^{intra}|+1}$, $p_i^E = \frac{p_i + \sum_{a_j \in N_i^{intra}} p_j}{|N_i^{intra}|+1}$.

After recording this estimation, we immediately notice that each boundary agent a_i maintains two types of load and capacity contents: its own actual content $\langle L_i, p_i \rangle$ and the estimated one $\langle L_i^E, p_i^E \rangle$ of its community. A question is then raised: which content should the boundary agent provide to its neighbors? Here, we propose that when a_i's intra-community neighbor $a_j \in N_i^{intra}$ inquires its load and capacity status, a_i should provide its own actual status to a_j; alternatively, when an intercommunity neighbor $a_k \in N_i^{inter}$ requests a_i's load and capacity information, it should provide the estimated one to a_k. For the sake of clarity, we generalize the two types of contents $\langle L_i, p_i \rangle$ and $\langle L_i^E, p_i^E \rangle$ as $\langle \hat{L}_i, \hat{p}_i \rangle$, and we assume that each boundary agent can always provide accurate information to its neighbors.

Second, we classify a boundary agent's domain into different sub-domains according to the community attribute they represent. This classification can be generated by using Algorithm 2.

Algorithm 2. Classify a Boundary Agent's Domain (a_i,Ω_i)

```
/* $C(\Omega_i)$: the set of communities that $\Omega_i$ belongs to;*/
/* $\Omega_{i,C_j}(C_j \in C(\Omega_i))$: the set of agents that belong to the
   domain community $C_j$. */
1. For $\forall a_k \in \Omega_i$, do
2.    If $\exists C_j \in C(\Omega_i)$: $com(a_k) = C_j$, then
3.        $\Omega_{i,C_j} \leftarrow \Omega_{i,C_j} \bigcup a_k$.
4.    Else
5.        $C(\Omega_i) \leftarrow C(\Omega_i) \bigcup com(a_k)$,
6.        $\Omega_{i,com(a_k)} \leftarrow \Omega_{i,com(a_k)} \bigcup a_k$.
7. End for
8. Return $C(\Omega_i)$ and $\Omega_{i,C_j}(\forall C_j \in C(\Omega_i))$.
```

Let there be a boundary agent a_i. For each domain community $C_j \in C(\Omega_i)$ of a_i, the average load of community C_j is $\bar{L}_{i,C_j} = \frac{\sum_{a_k \in \Omega_{i,C_j}} \hat{L}_k}{|\Omega_{i,C_j}|}$, the average capacity of community C_j is $\bar{p}_{i,C_j} = \frac{\sum_{a_k \in \Omega_{i,C_j}} \hat{p}_k}{|\Omega_{i,C_j}|}$, and the average utilization of community C_j is $\bar{u}_{i,C_j} = \frac{\bar{L}_{i,C_j}}{\bar{p}_{i,C_j}}$. The average utilization of all the domain communities $C(\Omega_i)$, $\bar{u}_{i,C(\Omega_i)}$, from the community perspective is:

$$\bar{u}_{i,C(\Omega_i)} = \sum_{C_j \in C(\Omega_i)} \bar{L}_{i,C_j} \Big/ \sum_{C_j \in C(\Omega_i)} \bar{p}_{i,C_j} \quad (7)$$

As discussed above, a boundary agent should not only balance the load within its community but also be responsible for migrating its community's excess load to the neighbor communities. Thus, the load status of a boundary agent should be detected from two perspectives: the load status from the intra-community perspective and that from the intercommunity perspective.

Definition 4.8 (Load status of boundary agent). *A boundary agent a_i is denoted:*

- *overloaded in the intercommunity domain if its intra-community average utilization ($\bar{u}_{i,com(a_i)}$) exceeds its domain communities' average utilization ($\bar{u}_{i,C(\Omega_i)}$);*
- *overloaded in the intra-community domain if its own utilization (u_i) exceeds its intra-community average utilization.*

The behaviors of the boundary agent a_i during load balancing can then be described as:

- *Intercommunity load balancing*: a_i checks whether it is overloaded in the intercommunity domain or not; if yes, it performs load balancing between communities.
- *Intra-community load balancing*: a_i compares its utilization with its intra-community average utilization and decides whether to balance the load within its community.

Next, we will describe the intercommunity and intra-community load balancing mechanisms for the boundary agent a_i in detail.

1) ***Intercommunity load balancing***

It is worth noting that if a_i performs load balancing with its intra-community neighbors, it makes no contribution to the intercommunity load fairness. Thus, we modify a_i's intercommunity load balancing strategy space S_i^{inter} as:

$$S_i^{inter} = \{s_i(a_j, t_k) | a_j \in N_i^{inter} \wedge t_k \in T(a_i)\}.$$

Definition 4.9 (Intercommunity load balancing benefit). *The benefit to a_i under intercommunity load balance strategy $s_i(a_j,t_k) \in S_i^{inter}$, $g_i^{inter}(s_i)$, is defined as: $g_i^{inter}(s_i) = 1 - SD^*_{i,C(\Omega_i)}/SD_{i,C(\Omega_i)}$, where $SD_{i,C(\Omega_i)}$ is the utilization standard deviation of $C(\Omega_i)$ before load balancing from the community perspective, i.e.,*

$$SD_{i,C(\Omega_i)} = \Big(\sum_{C_j \in C(\Omega_i)} (\bar{u}_{i,C_j} - \bar{u}_{i,C(\Omega_i)})^2 / |C(\Omega_i)| \Big)^{\frac{1}{2}} \quad (8)$$

and $SD^*_{i,C(\Omega_i)}$ is the utilization standard deviation of $C(\Omega_i)$ after load balancing.

The net profit of an intercommunity load balance strategy $s_i(a_j,t_k) \in S_i^{inter}$, *net-pro*$_i^{inter}(s_i)$, then is defined as:

$$net\text{–}pro_i^{inter}(s_i) = \beta g_i^{inter}(s_i) - w(a_i,a_j) \cdot l(t_k) \quad (9)$$

The coefficient $\beta (\beta > 0)$ is the influence factor of the intercommunity load balance benefit.

2) ***Intra-community load balancing***

After certain boundary agent a_j unloads its community's excess loads to its intercommunity neighbor $a_i \in N_j^{inter}$, it is also essential for a_i to apportion the excess loads to its intra-community neighbors. Agent a_i's intra-community load balancing strategy space is modified as:

$$S_i^{intra} = \{s_i(a_j,t_k) | a_j \in N_i^{intra} \wedge t_k \in T(a_i)\}.$$

Definition 4.10 (Intra-community load balancing benefit). *The benefit to a_i under the intra-community load balance strategy $s_i(a_j,t_k) \in S_i^{intra}$, $g_i^{intra}(s_i)$, is defined as: $g_i^{intra}(s_i) = 1 - SD^*_{i,com(a_i)}/SD_{i,com(a_i)}$, where $SD_{i,com(a_i)}$ is the utilization standard deviation of $\Omega_{i,com(a_i)}$ before load balancing from a_i's intra-community perspective, i.e.,*

$$SD_{i,com(a_i)} = \Big(\sum_{a_j \in \Omega_{i,com(a_i)}} (u_j - \bar{u}_{i,com(a_i)})^2 / |\Omega_{i,com(a_i)}| \Big)^{\frac{1}{2}} \quad (10)$$

and $SD^*_{i,com(a_i)}$ is the utilization standard deviation of $\Omega_{i,com(a_i)}$ after load balancing.

The net profit of an intra-community load balance strategy $s_i(a_j,t_k) \in S_i^{intra}$, *net-pro*$_i^{intra}(s_i)$, is given by:

$$net\text{–}pro_i^{intra}(s_i) = \alpha g_i^{intra}(s_i) - w(a_i,a_j) \cdot l(t_k) \quad (11)$$

The influence factor α of the intra-community load balancing benefit is similar to that described in Definition 4.6. Finally, a formal description of the load balancing procedure for the boundary agent can be seen in Algorithm 3.

Algorithm 3. Boundary Load Balancing Algorithm (a_i,Q)

```
1. Set $b = 0, c = 0$; compute $u_i$, $\bar{u}_{i,com(a_i)}$, and $\bar{u}_{i,C(\Omega_i)}$.
2. While ($\bar{u}_{i,com(a_i)} > \bar{u}_{i,C(\Omega_i)}$ && $T(a_i) \neq \emptyset$ && $b \neq 1$) do
3.     Select the optimal strategy $s_i^*(a_j,t_k)$ by Equation (9).
4.     If ($g_i^{inter}(s_i^*) > 0$), then
5.         Execute $s_i^*$; $Q = Q + w(a_i,a_j) \cdot l(t_k)$.
6.     Else $b$=1.
7. End While
8. While ($u_i > \bar{u}_{i,com(a_i)}$ && $c \neq 1$) do
9.     Select the optimal strategy $s_i^*(a_j,t_k)$ by Equation (11).
10.    If ($g_i^{intra}(s_i^*) > 0$), then
11.        Execute $s_i^*$; $Q = Q + w(a_i,a_j) \cdot l(t_k)$.
12.    Else $c$=1.
13. End While
```

C. Theoretical analysis

In our simulated distributed SN-MASs, each agent independently collects its domain information, decides whether to perform load balancing and selects the optimal load balance process to execute. An agent terminates load balancing if its optimal load balance strategy is with negative benefit value, and the system terminates if and only if all agents terminate. It follows from this termination condition that a question is raised: Does the load balancing converge to the optimal equilibrium state? Next, we will prove that our load balancing mechanism always converge to the optimal equilibrium state in certain scenarios.

Theorem 4.1. *Suppose that all agents have the same capacity, i.e., $p_1 = \ldots = p_m = p$, and the minimum load of a task is λ. Then the system takes at most $O(\Pi^2/\lambda^2)$ steps to reach the* *optimal load balance state, where Π is the sum loads of all the tasks in the system, i.e., $\Pi = \sum_{t_k \in T} l(t_k)$.*

Proof. For the sake of the proof, we use the metric utilization variance D instead of the utilization standard deviation SD, where $D = SD^2$. The proof is divided into two parts: *i)* we prove that if an agent executes a load balance process, there will be a substantial reduction in system utilization variance; *ii)* we prove that the difference between system utilization variance in the initial state and that in the optimal state is not larger than a finite value. As discussed above, agents can be categorized into internal and boundary cases.

Internal case: Assume that an internal overloaded agent a_i migrates its local task t_k to its under-loaded neighbor a_j at time t. From a_i's local domain perspective, we have:

$$\begin{aligned} D_i - D_i^* &= \big((u_i - \bar{u}_i)^2 + (u_j - \bar{u}_i)^2 - (u_i^* - \bar{u}_i)^2 - (u_j^* - \bar{u}_i)^2\big)/|\Omega_i| \\ &= l(t_k)\big(u_i + u_i^* - u_j - u_j^*\big)/\big(|\Omega_i| p\big) \\ &= 2l(t_k)\big(u_i - u_j - l(t_k)/p\big)/\big(|\Omega_i| p\big) \end{aligned} \tag{12}$$

where D_i and D_i^* are the utilization variances of a_i's domain Ω_i before and after load balancing. It follows from the constraint that the benefit of a load balance process must be positive, and the assumption that the minimum load of a task is λ, we can derive:

$$\left.\begin{aligned} l(t_k) \geq \lambda \\ \big(u_i - u_j - l(t_k)/p\big) \geq \lambda/p \end{aligned}\right\} \Rightarrow D_i - D_i^* \geq 2\lambda^2/(|\Omega_i| p^2) \tag{13}$$

Alternatively, from the system global perspective, we have:

$$\begin{aligned} D - D^* &= \big((u_i - \bar{u})^2 + (u_j - \bar{u})^2 - (u_i^* - \bar{u})^2 - (u_j^* - \bar{u})^2\big)/m \\ &= l(t_k)(u_i + u_i^* - u_j - u_j^*)/(mp) \\ &= (D_i - D_i^*)|\Omega_i|/m \geq 2\lambda^2/(mp^2) \end{aligned} \tag{14}$$

where m is the number of agents, $\bar{u}$ is the system's average utilization, D and D^* are the system's utilization variances before and after load balancing. Up to this point, we have proved that an internal load balance process reduces the system utilization variance by at least $2\lambda^2/(mp^2)$.

Boundary case: It follows from remark 4.2 that the boundary agent gains more benefit than the internal agent if it migrates its community's excess load to its neighbor communities. On the other hand, when the boundary agent balances load within its community, the gained benefit is equivalent to that of the internal case. Therefore, we can determine that the boundary load balance process also reduce the system utilization by at least $2\lambda^2/(mp^2)$.

Now we will prove *ii)*. Notice that the worst case in the initial state is that the entire system load is distributed to a single agent, where the system utilization variance D_{ini} is:

$$\begin{aligned} D_{ini} &= \big[\big(\Pi/p - \Pi/(pm)\big)^2 + (m-1)\big(\Pi/(pm)\big)^2\big]/m \\ &= (m-1)\Pi^2/(m^2p^2) \end{aligned} \tag{15}$$

On the other hand, in the optimal load balance state, the entire system load Π is distributed among agents evenly, i.e., each agent has the same utilization: u_1=...=u_m=$\Pi/(mp)$, where we have the system utilization variance D_{opt}=0. The difference between D_{ini} and D_{opt} then is $D_{ini} - D_{opt} \leq D_{ini}$=$(m-1)\Pi^2/(m^2p^2)$. Finally, to reach the optimal balance state, the system needs at most $(D_{ini} - D_{opt})/(D - D^*) \leq (m-1)\Pi^2/(2m\lambda^2)$ steps. Now, we can conclude that the system will converge to the optimal load balance state in $O(\Pi^2/\lambda^2)$ steps. $\square$

In this study, because of the heterogeneity of agents' capacities, the local load balance process might not always alleviate the system global load unfairness (sometimes, the process may even aggravate the unfairness). However, it will not prevent the system converging to an approximate optimal load fairness state (This will be validated in the experiments).

V. Experiments

A. Experiment setting

We apply our load balancing approach to a set of artificial SN-MASs with explicit communities which were used in [13]. Each SN-MAS consists of 1024 agents partitioned into 64 communities, each with 16 agents. Next, we connect the agents randomly, with the probability p_{intra} for agents to connect to their intra-community members, and p_{inter} for agents to connect to the intercommunity individuals. The probabilities p_{intra} and p_{inter} are chosen to create a network with a community structure (here, we set p_{inra}/p_{inter}=4). Finally, we assign weight (migration cost) to the connections: intra-community connections are given the weight of $U(0,1)$ ($U(a,b)$ returns a value distributed over the interval (a,b] uniformly), while those between communities are set to $U(2,3)$.

The capacity of each agent is given as $U(0,20)$ and we assume that there are 10000 tasks waiting to process in the system. We set the system loads associated with these tasks equal to the total capacities of agents (i.e., the system's average utilization $\bar{u}$=1) and distribute these system loads to tasks uniformly. Next, we distribute these tasks over agents as follows: *1) Simulating the loads of communities:* we use the Gauss distribution $N(\mu, \sigma^2)$ to simulate the utilization of each community. The parameter μ represents the average utilization of all communities, and σ represents the standard deviation of community utilization (here, we set μ=σ=1). The load that each community undertakes is given by its utilization multiply its capacity. *2) Distributing tasks to communities:* we distribute tasks to communities randomly (tasks would not be distributed to a certain community if the total loads of that community reach the bound value simulated in step *1)*). When a task is distributed to a community, the location (agent) in that community is randomly selected. Finally, the influence factors α and β, providing for the benefit involved in calculating the net profit for the internal and the boundary agents, are set to 10 and 20, respectively.

We compare our migration cost-sensitive load balancing approach with three migration cost-ignorant algorithms: the load-based random algorithm, the benefit-based greedy algorithm, and the agent-based probability algorithm.

- **The load-based random model (Random model).** An overloaded agent selects a task from its local task set randomly, and migrates this task to the neighbor agent that has the lightest load [16].
- **The benefit-based greedy model (Greedy model).** An agent executes the optimal load balance process that has the maximum benefit value [3].
- **The agent-based probability model (Probability model).** A task(agent) probabilistically decides to join or leave a node based on the number of tasks queuing at that node [4].

We evaluate the performance of these methods by the system global load balance benefit (E) and the migration cost (Q). We are charged the migration cost each time a task is migrated between agents. The system global load balance benefit is computed as follows: let the initial system utilization standard deviation be $SD(0)$, and as the load balancing progresses, the system utilization standard deviation becomes $SD(t)$ at time t. Then, we have the system load balance benefit, $E(t)$ at time t is: $E(t)=1-SD(t)/SD(0)$.

B. Simulation results

Figure 2 shows the community utilizations before and after load balancing. The x-axis represents the community identity, and the y-axis is the community's utilization. From Figure 2, we can determine that our load balancing approach can reduce the imbalance of community utilizations: having heavily-burdened communities send their excess loads out to alleviate their high load level (e.g., the utilization of community 14 is 1.5 before load balancing, it drops down to 1.0 after load balancing) and the light-burdened communities receive these excess loads to improve the system performance (e.g., the utilization of community 48 is 0 before load balancing, it rises up to 0.9 after load balancing).

Figure 3 shows the performances of the community-aware (Our model), load-based random (Random model), benefit-based greedy (Greedy model) and agent-based probability (Probability model) load balancing algorithms with respect to the system benefit. The x-axis denotes the system time and the y-axis represents the system benefit. From the experimental results in Figure 3, we have the following observations: 1) In the early stages, the Greedy, the Probability and the Random models outperform our model. This can be explained by the fact that in the earlier stages, the load balance decision-making by the boundary agent of our model attempts to minimize the load unfairness of communities, which might make no contribution to system load fairness from a single agent's perspective. 2) As the load balancing proceeds, our model gains a higher benefit than the others and can eventually reach as high as 0.96. The reason is that the agents in the community-ignorant models (i.e., Greedy, Probability and Random) only focus on minimizing load unfairness from their own local domain perspectives. Obviously, this will lead the system to converge to a locally balance state. In contrast to these models, our community-aware model does not only balance the load within local domain (internal load balancing) but also reduces the load unfairness from community's perspective (boundary load balancing), thereby making the system converge to a global load fairness state. 3) Our model converges to the stable state faster than other models. This can be explained as follows: when the boundary agents have balanced the loads of communities, it will be easy for the internal agents to converge to the local intra-community fair state due to the dense interactions within communities.

Figure 4 evaluates the migration costs of these algorithms. From this figure, we can observe that the overhead migration cost of our migration cost-sensitive model is less than those of the Probability, the Random and the Greedy models. This is because these migration cost-ignorant approaches only

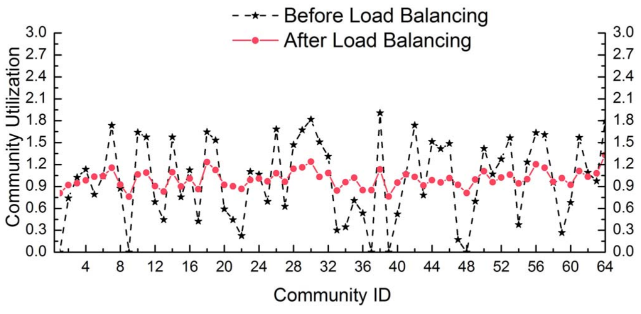

Figure 2: Community utilization comparison between before load balancing and after load balancing.

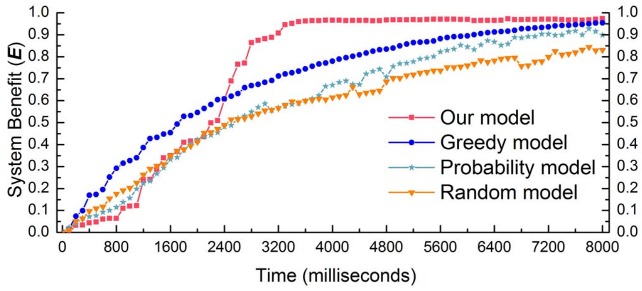

Figure 3: Performance comparison of load balancing algorithms in terms of system benefit E.

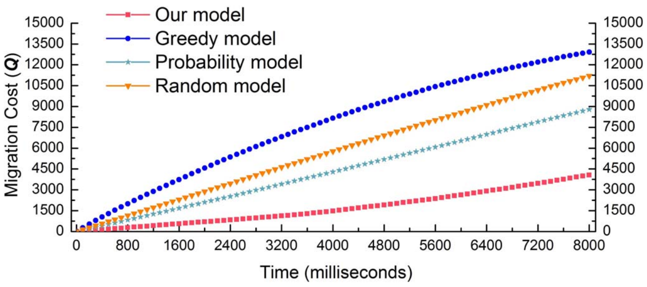

Figure 4: Performance comparison of load balancing algorithms in terms of movement cost Q.

Table I: Results of different α and β

Case	α	β	E	Q
1	2	4	0.9084	2146
2	4	8	0.9149	2348
3	6	12	0.9195	2845
4	8	16	0.9207	3385
5	10	20	0.9249	4022
6	12	24	0.9286	4672
7	14	28	0.9291	5255
8	16	32	0.9301	5726
9	18	36	0.9374	6302
10	20	40	0.9337	6805

aim to minimize the load imbalance of agents and do not consider the migration cost at all, which definitely will incur tremendous overhead migration costs.

Finally, we investigate the effect of the influence factors α and β, providing for the benefit involved in calculating the net profit for the internal and boundary agents. We vary the value of α from 2 to 20 with intervals of 2, and vary the value of β from 4 to 40 with intervals of 4. The results of these test cases are shown in Table I. From Table I, we can find that when α and β become larger, the gained benefit increases slightly as well, while the overhead migration cost increases rapidly. This can be explained by the fact that the larger α and β are, the greater influence of the benefit of a load balance process. Consequently, the agents are more likely to execute the process that has a larger benefit value, ignoring the overhead migration cost. Therefore, the coefficients α and β should be set properly to make a tradeoff between the system benefit and the migration cost.

VI. Conclusion

In this paper, we devise a migration cost-sensitive load balancing mechanism to address the load balancing problem for SN-MASs with communities. In our mechanism, the internal agents execute the optimal load balance process that has the maximum net profit value, thereby alleviating system local load unfairness and reducing the migration cost. A boundary agent migrates its community's excess loads to the neighbor communities first, and later performs load balancing within its community, thereby making the system converge to the global load fairness state. The simulation results demonstrate that there are three advantages of our load balancing mechanism. First, the mechanism makes the system converge to an approximately perfect fair load distribution state. Second, the mechanism converges to the steady state faster than other load balancing models that ignore the community structure. Third, the mechanism reduces the overhead migration cost significantly compared with the previous migration cost-ignorant approaches.

In this paper, we make the restrictive assumption that tasks are independent. However, in real-world applications, tasks might be interdependent; that is, there exist social communications among tasks [3]. This property will involve significant communication load for the agents where the tasks are queuing. Therefore, in the future, we will extend our load balancing model such that it takes the communication load into consideration.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (No.61170164), the Funds for Distinguished Young Scholars of Jiangsu Province of China (No.BK2012020), and the Program for Distinguished Talents of Six Domains in Jiangsu Province of China (No.2011-DZ023).

References

[1] W. Chen, Z. Liu, X. Sun, and Y. Wang, "Community Detection in Social Networks Through Community Formation Games", *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence(IJCAI-11)*, pp.2576-2581, Barcelona, Spain, July 16-22, 2011.

[2] Y. Jiang, Y. Zhou, and W. Wang, "Task Allocation for Undependable Multiagent Systems in Social Networks", *IEEE Transactions on Parallel and Distributed Systems*, 24(8): 1671-1681, 2013.

[3] K.P. Chow and Y.K. Kwork, "On Load Balancing for Distributed Multiagent Computing", *IEEE Transactions on Parallel and Distributed Systems*, 13(8): 787-801, 2002.

[4] J. Liu, X. Jin, and Y. Wang, "Agent-based Load Balancing on Homogeneous Minigrids: Macroscopic Modeling and Characterization", *IEEE Transactions on Parallel and Distributed Systems*, 16(7): 586-598, 2005.

[5] C.C. Hui and S.T. Chanson, "Hydrodynamic Load Balancing", *IEEE Transactions on Parallel and Distributed Systems*, 10(11): 1118-1137, 1999.

[6] Y. Jiang and J. Jiang, "Contextual Resource Negotiation-Based Task Allocation and Load Balancing in Complex Software Systems", *IEEE Transactions on Parallel and Distributed Systems*, 20(5): 641-653, 2009.

[7] Y. Jiang and Z. Li, "Locality-Sensitive Task Allocation and Load Balancing in Networked Multiagent Systems: Talent Versus Centrality", *Journal of Parallel and Distributed Computing*, 71(6): 822-836, 2011.

[8] B. Godfrey, K. Lakshminarayanan, S. Surana and R. Karp, "Load Balancing in Dynamic Structured P2P Systems", *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-04)*, pp.2253-2262, Hong Kong, March 7-11, 2004.

[9] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-based P2P Systems", *IEEE Transactions on Parallel and Distributed Systems*, 16(4): 349-361, 2005.

[10] J. Hu and R. Klefstad, "Decentralized Load Balancing on Unstructured Peer-2-Peer Computing Grids", *The Fifth IEEE International Symposium on Network Computing and Applications*, pp.247-250, Massachusetts, USA, July 24-26, 2006.

[11] M. Girvan and M.E.J. Newman, "Community Structure in Social and Biological Networks", *Proceedings of the National Academy of Sciences of the United States of America*, 99(12): 7821-7826, 2002.

[12] A. Barrat, M. Barthlemy, R.P. Satorras, and A. Vespignani, "The Architecture of Complex Weighted Networks", *Proceedings of the National Academy of Sciences of the United States of America*, 101(11): 3747-3752, 2004.

[13] M.E.J. Newman, "Analysis of Weighted Networks", *Physical Review E*, 70(5): 056131, 2004.

[14] S. Penmatsa and A.T. Chronopoulos, "Game-Theoretic Static Load Balancing for Distributed Systems", *Journal of Parallel and Distributed Computing*, 71(4): 537-555, 2011.

[15] S. Kraus, O. Shehory, and G. Taase, "Coalition Formation with Uncertain Heterogeneous Information", *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pp.1-8, Melbourne, Australia, July 14-18, 2003.

[16] A. Chavez, A. Moukas, and P. Maes, "*Challenger*: A Multi-Agent System for Distributed Resource Allocation", *Proceedings of the First International Conference on Autonomous Agents (ICAA-97)*, pp.323-331, California, USA, February 05-08, 1997.