PAPER

A Biologically Inspired Self-Adaptation of Replica Density Control

Tomoko IZUMI[†a)], **Taisuke IZUMI**[††], **Fukuhito OOSHITA**[†††], ***Members***, **Hirotsugu KAKUGAWA**[†††], ***Nonmember***, ***and*** **Toshimitsu MASUZAWA**[†††], ***Member***

SUMMARY Biologically-inspired approaches are one of the most promising approaches to realize highly-adaptive distributed systems. Biological systems inherently have self-∗ properties, such as self-stabilization, self-adaptation, self-configuration, self-optimization and self-healing. Thus, the application of biological systems into distributed systems has attracted a lot of attention recently. In this paper, we present one successful result of bio-inspired approach: we propose distributed algorithms for resource replication inspired by the single species population model. Resource replication is a crucial technique for improving system performance of distributed applications with shared resources. In systems using resource replication, generally, a larger number of replicas lead to shorter time to reach a replica of a requested resource but consume more storage of the hosts. Therefore, it is indispensable to adjust the number of replicas appropriately for the resource sharing application. This paper considers the problem for controlling the densities of replicas adaptively in dynamic networks and proposes two bio-inspired distributed algorithms for the problem. In the first algorithm, we try to control the replica density for a single resource. However, in a system where multiple resources coexist, the algorithm needs high network cost and the exact knowledge at each node about all resources in the network. In the second algorithm, the densities of all resources are controlled by the single algorithm without high network cost and the exact knowledge about all resources. This paper shows by simulations that these two algorithms realize self-adaptation of the replica density in dynamic networks.

key words: *replica density control, resource replication, bio-inspired approach, single species population model*

1. Introduction

Continuous and significant increase in scale, dynamics and diversity of network environments requires highly-adaptive distributed systems. Thus, autonomic distributed systems with self-∗ properties, such as self-stabilization, self-adaptation, self-configuration, self-optimization and self-healing, are attracting widespread attention from researchers and engineers in the fields of distributed systems.

Biological systems inherently have self-∗ properties to realize environmental adaptation. Thus, several biologically-inspired approaches have succeeded in realizing highly adaptive distributed systems. Successful projects include Bio-Networking project [1] and Anthill project [2]. These projects adopt biologically-inspired approaches to provide a highly adaptive platform for mobile-agent-based computing [3], [4].

This paper presents one successful result of bio-inspired approach in dynamic distributed systems: we propose distributed algorithms for *resource replication* inspired by a biological system. Resource replication is a crucial technique for improving performance and availability of resource sharing systems, which are one of the most fundamental distributed applications (a well-known example is file sharing on peer-to-peer networks [5], [6]). In resource sharing systems using replication, replicas of an original resource are distributed over the network so that each user can get a requested resource by accessing a nearby replica. Resource replication can reduce communication latency and consumption of network bandwidth and can also improve availability of the resources even when some of the replicas are unavailable.

In systems using resource replication, generally, a larger number of replicas lead to shorter time to reach a replica of a requested resource, but consume more storage of the hosts. Thus, it is indispensable to adjust the number of replicas appropriately for the system. It is natural to consider that the system, where each host provides a constant amount of storage for keeping replicas, should keep as many replicas as possible to improve system performance. The total amount of storage in the system depends on the network size. In this paper, we consider control algorithms on application layer (e.g., on overlay in a peer-to-peer network) to adapt the number of replicas depending on the amount of storage at each host and the current network size (i.e., the number of hosts). By applying an adaptive algorithm for the controlling the number of replicas, the system can guarantee QoS to a certain extent regardless of the network size. One such example is resource searching protocol PWQS, which is a quorum-based probabilistic protocol on peer-to-peer networks [7]. In PWQS, replicas are dispersed randomly in a network, and a user searches a requested resource by accessing hosts randomly. While PWQS requires replicas of each resource in numbers proportional to the network size to attain good performance, an algorithm for the control of the number of replicas is not proposed in [7].

However, in dynamic networks such as peer-to-peer networks, an appropriate number of replicas for the applica-

Manuscript received July 24, 2008.
Manuscript revised December 19, 2008.

[†]The author is with Center for Social Contribution and Collaboration, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan.
[††]The author is with the Graduate School of Engineering, Nagoya Institute of Technology, Nagoya-shi, 466–8555 Japan.
[†††]The authors are with the Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan.
a) E-mail: izumi.tomoko@nitech.ac.jp

DOI: 10.1587/transinf.E92.D.1125

tion changes with time, since network size varies with time. When a special host does centralized control of the number of replicas, it should compute global information repeatedly to adapt to dynamic changes of the network, such as the current network size and the number of replicas. The computation requires much greater cost since the host must collect state of all hosts (e.g., replicas each host has). So, it is unrealistic to assume that each node knows such global information. Moreover, in terms of fault tolerance, a system should not have such a special host. Therefore, we focus distributed algorithms to realize adaptable control of the number of replicas in dynamic networks without a special host and global information.

Control of the number of replicas and replica allocation, which determines the node each replica is allocated to, improves the system performance. We focus only on control of the number of replicas in this paper, meanwhile many other researchers focus on replica allocation [8], [9]. Our algorithms can be combined to these allocation protocols since the algorithms can control the number of replicas independently from replicas' allocation. For the control of the number of replicas, some papers have proposed centralized and distributed algorithms [10], [11]. These algorithms control the number of replicas depending on a request frequency distribution of resources. As stated above, in this paper, we focus on the amount of storage for keeping replicas in a network. That is, we first propose a problem for the control of the replica density adaptively in response to change of network size.

Contribution of this paper. In this paper, we first formulate *the replica density control problem* in dynamic networks, and present biologically-inspired algorithms for the problem. The replica control problem requires us to adapt the total number of replicas to a given constant fraction of the current network size and to adapt the number of replicas of each resource equally.

Our algorithms borrow an idea from *the single species population model*, which is a well-known population ecology model. This model considers population of a single species in an environment such that individuals of the species can survive by consuming food supplied by the environment. The model is formulated by *the logistic equation* and shows that the population automatically converges to and stabilizes at some number depending on the amount of supplied food.

In the proposed algorithms, replicas are regarded as individuals of a single species. To supply food to replicas, we use mobile agents created by nodes. The first algorithm focuses on controlling the replica density of a single resource by supplying an appropriate amount of food. The simulation results show that the first algorithm can adequately adjust the replica density. In the system where multiple resources coexist, we can control the replica density of each resource by applying this algorithm to each resource: replicas of resource i are supplied food by agents that have food for resource i.

However, since the first algorithm controls the replica density of each resource independently, the number of agents on the network is proportional to the number of resources. Moreover, it is necessary for each node to have the exact knowledge about all resources existing in the network. To solve these problems, we propose the second algorithm in which densities of all resources are controlled without a great number of agents and the exact knowledge about all resources. In the second algorithm, every replica can eat any food supplied by agents. Since the technique for supplying food fairly to replicas of every resource is needed, we limit the amount of food supplied to replicas of each resource. The simulation results show that the second algorithm also realizes self-adaptation of the replica density in dynamic networks.

We have already proposed the algorithm to control the number of objects in dynamic networks through the same approach in [12], [13]. In [13], we proposed an algorithm for control the replica density of only one resource while we try to control the densities of multiple resources at the same time in this paper. In [12], we tried to control the mobile agent population. The difference between the control algorithms of replicas and mobile agents is that the target objects are static or dynamic. Mobile agents can migrate between nodes and eat food for itself, but replicas cannot migrate in the network.

The rest of this paper is organized as follows. In Sect. 2, we present the model of distributed systems, and define the replica density control problem. In Sect. 3, we explain about the single species population model, and in Sect. 4, we propose the without a special host and global information without a special host and global information first distributed algorithm for the problem and show its simulation results. The second algorithm is presented and simulated in Sect. 5. Section 6 concludes the paper.

2. Preliminaries

2.1 System Models

Dynamic networks. In this paper, we consider a *dynamic network* such that its node set and its link set vary with time. To define dynamic networks, we introduce discrete time and assume that each time is denoted by a non-negative integer in a natural way: time 0 denotes the initial time, time 1 denotes the time immediately following time 0 and so on.

Formally, a dynamic network at time t is denoted by $N(t) = (V(t), E(t))$, where $V(t)$ and $E(t)$ are respectively the node set and the link set at time t. A link in $E(t)$ connects two distinct nodes in $V(t)$ and a link between nodes u and v is denoted by e_{uv} or e_{vu}. We also use the following notations to represent the numbers of nodes and edges at time t: $n(t) = |V(t)|$ and $e(t) = |E(t)|$.

Mobile agent systems. A *mobile agent* is an autonomous program that can migrate from one node to another on the network [14], [15]. In a dynamic network, an agent on a

node $u \in V(t)$ at time t can start migrating to a node $v \in V(t)$ only when link e_{uv} is contained in $E(t)$. The agent reaches v at time $t + \Delta$ only when the link e_{uv} remains existing during the period from t to $t + \Delta$, where Δ is an integer representing *migration delay* between the nodes. The agent migrating from u to v is removed from the network when the link e_{uv} disappears during the period from t to $t + \Delta$.

Each of nodes and agents has a *local clock* that runs at the same rate as the global time. However, we make no assumption on the local clock values: the difference between the local clock values in the system is unbounded.

An agent and a node can interact with each other by executing operations: an agent p on a node u can change its state and the state of u depending on the current states of p and u, and u can change its state and the states of the agents residing on u depending on the current states of u and the agents. Besides the above operations, each agent can execute operations to kill itself and each node can also execute operations to create new agents.

When agents reside on a node, the agents and the node have operations they can execute. For execution semantics, we assume that the agents and the node execute their operations sequentially in an arbitrary order. We also assume that the time required to execute the operations can be ignored, that is, we consider all the operations are executed sequentially but at an instant time.

2.2 Replica Density Control Problem

In this paper, we consider the *replica density control problem*. Resources are objects shared by the nodes on the network: files, documents, and so on. Replicas are copies of an original resource. Each node has zero or more replicas. We distinguish an original resource from its replicas: an original resource is allocated on its original node and is never deleted unless the original node decides to delete the resource, while a replica of the original resource can be deleted. Each node v can create same replicas from a replica or an original resource on v and can delete replicas on v. Let $r(t)$ be the total number of replicas (including original resources) and $r_i(t)$ be the number of replicas of a resource i (including the original resource i) on the network $N(t)$ at time t. The problem is defined as follows.

Definition 2.1
The goal of **the replica density control problem** *is to adjust the total number of replicas $r(t)$ and the number of replicas $r_i(t)$ of each resource i at time t to satisfy the following equations for a given constant δ.*

$$r(t) = \delta \cdot n(t)$$
$$r_i(t) = r_j(t) \qquad \forall i, j$$

We assume that the constant δ is initially given to every node. The parameter δ represents the amount of storage each node can assign to keeping replicas. The expression $\delta \cdot n(t)$ is the total amount of storage on the network at time t. If nodes can maintain many replicas in their storage then δ should be set to a large value. In this case, since a large amount of storage is provided, the system can achieve good performance by creating a large number of replicas. The first equality represents the adaptation of the current total number of replicas to the current total amount of storage on the network. That is, the more nodes exist on the network or the more amount of storage is provided, the more replicas can be kept on the whole network. The second equality represents the restriction on the number of replicas of each resource. That is, when there are many resources on the network, each resource cannot be kept many its replicas since the replicas of the other resources also consume much storage of nodes. On the other hand, when there are not many resources on the network, each resource can be kept many its replicas.

In some applications, a control of replica density depending on popularity of resources may be desired. Although our algorithms try to keep the number of replicas of each resource equally, they can control the number of replicas of each resource depending on its popularity by adding some preprocessing. We discuss the preprocessing in Conclusion.

We consider a distributed system such that replicas are distributed over the network and nodes can leave or join the networks at any time. In such environment, it is obviously impossible to keep satisfying the above equation all the time. Thus, our goal is to propose distributed algorithms that realize convergence to and stability at the target number.

3. Single Species Population Model

In this section, we introduce the *single species population model*. The model formulates the population growth of the species in the environment, and shows that the population (i.e., the number of individuals) in the environment automatically converges to and stabilizes at some number depending on the amount of food supplied by the environment.

Each individual of the species periodically needs to take a specific amount of food to survive. That is, if an individual can take the specific amount of food then it can survive. Otherwise, it dies. Moreover, in the case that an individual can take a sufficient amount of extra food, then it generates progeny. Consequently, the followings hold: the shortage of supplied food results in decrease in the population. Conversely, the excessive amount of food results in increase in the population.

The single species population model formulates the above phenomena. Let $p(t)$ be the population at time t. The single species population model indicates that the *population growth rate* at time t is represented by the following nonlinear first-order differential equation known as the *logistic equation* [16]:

$$\frac{\Delta p(t)}{\Delta t} = p(t) \cdot g(t) = p(t)(k \cdot f_a(t) - k \cdot f \cdot p(t)),$$

where $f_a(t)$ is the amount of food supplied by the environment at time t, f is the amount of food consumed by one

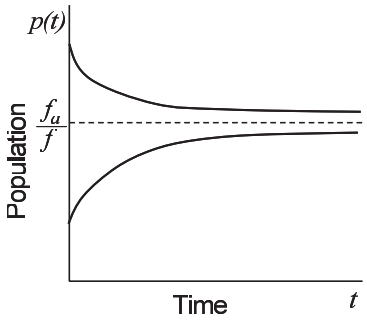


Fig. 1 Convergence to the equilibrium point in logistic equation.

individual to survive and k is a positive real constant. The function

$$g(t) = k(f_a(t) - f \cdot p(t))$$

in the above formula is called the *per capita growth rate* at time t.

The expression $f_a(t) - f \cdot p(t)$ represents the difference between the amounts of supplied food and consumed food. When the supplied food exceeds the consumed food, $g(t)$ takes a positive value proportional to the amount of the surplus food. A scarcity of the supplied food causes a negative value of $g(t)$ proportional to the shortage of the supplied food.

The logistic equation has two equilibrium points of the population size $p(t)$: $p(t) = 0$ and $p(t) = f_a(t)/f$. That is, the population remains unchanged, when the population size is at the equilibrium points. The equilibrium point $p(t) = f_a(t)/f$ represents the maximum population that the environment can keep, and is called the *carrying capacity* of the environment.

If the population is larger (resp. smaller) than the carrying capacity then the population decreases (resp. increases). Once the population reaches the carrying capacity, then it remains unchanged (see Fig. 1). Consequently, the single species population model implies that the population eventually converges to and stabilizes at the carrying capacity. Notice that the carrying capacity depends on the amount of food supplied by the environment.

4. Algorithm for Replica Density Control of a Single Resource

In this section, we propose an algorithm for the replica density control problem of a single resource.

4.1 Algorithm

In the algorithm, we introduce time interval of some constant length denoted by *CYCLE*. Behavior of each node and each replica can be divided into series of the time interval: each node supplies food every the time interval and decides by the food consumption of each replica whether the replica can survive to the next time interval or not. It should be noticed that the start time of the intervals at different nodes need not be synchronized. Moreover, the decisions about survival of different replicas on the same node are made at the different times.

Figure 2 and Fig. 3 show the detailed behaviors of a node and an agent in our algorithm. In the algorithm, we do not specify the allocation of replicas since our algorithm is independent of replicas' allocation.

The algorithm is simple: each node creates a new agent every *CYCLE* time units (i.e., at the beginning of each time interval). Each agent has a specific amount of food on the initial state and traverses the network with the food. In this paper, we adopt a *random walk* as migration pattern of an agent since it is adaptive and the simplest migration pattern without memory space of the agent. That is, each agent repeatedly executes the following actions: each agent stays at a node for a stay time, and then migrates to one of its neighboring nodes with equal probability. When an agent visits a node v, the agent feeds the replicas on v. The replica can survive to the next time interval if it can be fed a specific amount of food, denoted by *RF*, during the current time interval. The replica is deleted if it cannot be fed food of amount *RF* during the time interval. The original resource behaves the same as a replica: it consumes food just like replicas, but is not deleted even if it cannot be fed food of amount *RF*. When all the food the agent p carries is consumed, p kills itself (i.e., removes itself from the network).

In addition, each node creates a new replica of resource i if a replica i_h of resource i is fed surplus food of amount *RF*. This idea derives from the fact that the positive per capita growth $g(t)$ in the single species population model is proportional to the amount of surplus food. This scheme is realized in the following way: if all food an agent has is not consumed by replicas after *CYCLE* time units from its creation time, the agent stores the surplus food into variable *surplus_food* and continues a random walk. The amount of surplus food implies the number of replicas the system should create since agents supply adequate amount of food to keep an appropriate number of replicas. Thus, the surplus food should be consumed by replicas to create new replicas. In our algorithm, when an agent carrying surplus food visits a node v, the agent feeds replicas on node v with the surplus food. If the total amount of surplus food the replica, say replica i_h, is fed reaches *RF*, one new replica of resource i (i.e., a copy of replica i_h) is created. If the agent feeds all the surplus food, it kills itself.

Now, we consider the amount of food F that each agent should supply. Since each node creates one agent every *CYCLE* time units, the amount of food supplied on the whole network at time t can be approximately estimated to be $F \cdot n(t)$. The goal of the replica density control problem is to adjust the total number $r(t)$ of replicas to $\delta \cdot n(t)$. Remind that the single species population model shows that the number of individuals converges to and stabilizes at the carrying capacity $f_a(t)/f$. Thus, the algorithm tries to adjust $r(t)$ to $\delta \cdot n(t)$ by adjusting the carrying capacity to $\delta \cdot n(t)$. Since $f_a(t)$ corresponds to the total amount of supplied food on the whole network $n(t) \cdot F$ and f corresponds to the amount of

Behavior of node v

$time_v$: local clock time
/* its value automatically increases at the same rate as the global time */
$eat\_food(i_h)$: the amount of food that replica i_h of resource i has consumed in the current time interval
$eat\_surplus\_food(i_h)$: the amount of surplus food that replica i_h has consumed
$create\_time(i_h)$: creation time of replica i_h /* the time at which h is made */
RF : the amount of food to be consumed by a replica to survive
$CYCLE$: the time interval of behavior of the node and each replica

- at the beginning of each time interval of node v (i.e., when $time_v$ **mod** $CYCLE$ =0 holds)
  create one agent
- on agent p's arrival at node v
  for each replica i_h on node v
    if ($food_p > 0$) **then**
      if ($eat\_food(i_h) < RF$) **then**
        $y := \min\{RF - eat\_food(i_h), food_p\}$
        $eat\_food(i_h) := eat\_food(i_h) + y$
        $food_p := food_p - y$
      else /* the agent is carrying the surplus food */
        $y' := \min\{RF - eat\_surplus\_food(i_h), surplus\_food_p\}$
        $eat\_surplus\_food(i_h) := eat\_surplus\_food(i_h) + y'$
        $surplus\_food_p := surplus\_food_p - y'$
        if ($eat\_surplus\_food(i_h) = RF$) **then**
          create a new replica i_j of resource i (i.e., a copy of replica i_h)
          $create\_time(i_j) := time_v$
          $eat\_food(i_j) := 0$
          $eat\_surplus\_food(i_j) := 0$
          $eat\_surplus\_food(i_h) := 0$
- at the end of each time interval of each replica h
  (i.e., when $time_v - create\_time(i_h)$ **mod** $CYCLE$ =0 holds)
  if ($eat\_food(i_h) < RF$ and i_h is not an original resource i) **then** delete i_h
  else $eat\_food(i_h) := 0$ /*i_h survives into the next time interval */

Fig. 2 Behavior of node v in the first algorithm.

Behavior of agent p

$food_p$: the amount of food p carries /* $food_p > 0$ implies $surplus\_food_p = 0$ */
$surplus\_food_p$: the amount of surplus food p carries
/* $surplus\_food_p > 0$ implies $food_p = 0$ */
$time_p$: local clock time
/* its value automatically increases at the same rate as the global time */
RF : the amount of food to be consumed by a replica to survive
$CYCLE$: the time when the food the agent has changes to surplus food
δ : the target ratio between the number of replicas and the network size
/*p makes a random walk on the network */

- when p is created
  $food_p := \delta \cdot RF$
  $surplus\_food_p := 0.0$
  $time_p := 0$
- at the end of time interval of agent p (i.e., when $time_p = CYCLE$ holds)
  $surplus\_food_p := food_p$
  $food_p := 0.0$
- when all food is consumed (i.e., when ($food_p = 0.0 \wedge surplus\_food_p = 0.0$) holds)
  kill itself

Fig. 3 Behavior of agent p in the first algorithm.

food RF, the following equation should be satisfied:

$$\frac{f_a(t)}{f} = \frac{n(t) \cdot F}{RF} = \delta \cdot n(t).$$

From this equation, the amount of food F each agent should supply is determined to be $F = \delta \cdot RF$.

4.2 Simulation Results

In this subsection, we present simulation results to show that the proposed algorithm can adjust the replica density.

The following values are initialized randomly:

- the initial number and locations of agents
- the initial locations of replicas
- the initial values of the local clocks (i.e., $time_v$, $time_p(< CYCLE)$)
- the creation time of replicas on each node v (i.e., $create\_time(i_h)(< time_v)$)

The initial amounts of food that agents have (i.e., $food_p$) and replicas have fed on in the current time interval (i.e., $eat\_food(i_h)$) are set to the value based on the local clocks and creation times: $food_p = (1 - time_p/CYCLE) \cdot \delta RF$ and $eat\_food(i_h) = (create\_time(i_h)/CYCLE) \cdot RF$. The initial amounts of surplus food that agents have (i.e., $surplus\_food_p$) and the initial amounts of surplus food that replicas have fed (i.e., $eat\_surplus\_food(i_h)$)are set to 0.

Our algorithm can be combined to any allocation protocols since the algorithm can control the replica density independently from replicas' allocation. In the simulations, a new replica created by a node is allocated to the node selected randomly with probability proportional to their degrees. The random allocation based on nodes' degrees is one of the simplest allocations to guarantee search performance since a node with high degree is likely to be accessed by many nodes to search resources. The random allocation can be realized in a real system as follows: an agent with a replica traverses the network by a random walk during randomly long time units.

We present the simulation results for *random networks* and *scale-free networks*. Scale-free networks are a specific kind of networks whose degree distribution follow a power law. A scale-free network is said to be a realistic model of actual network structures [17], [18]. In the simulation, the stay time of an agent at a node is set to one time unit and the migration delay between any pair of neighboring nodes is set to two time units.

To show the adaptiveness of the proposed algorithm, we also show the difference ratio of the number of replicas: the ratio is defined by $|\delta \cdot n(t) - r(t)|/(\delta \cdot n(t))$ and represents the ratio of difference between the adjusted and the target numbers of replicas to the target number.

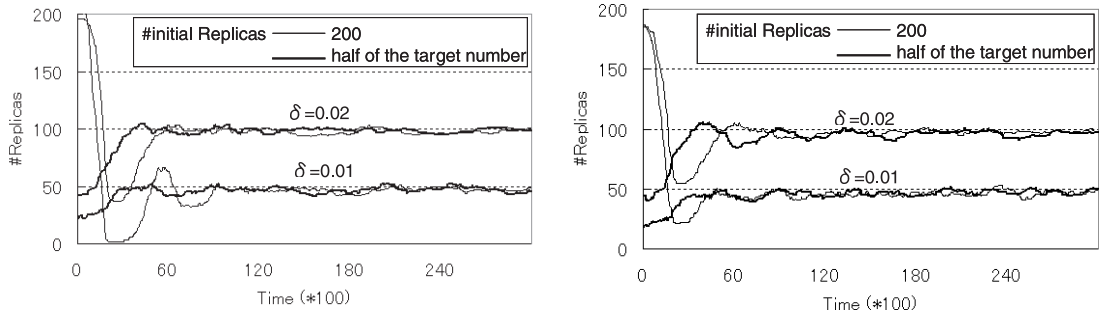
Simulation results for static networks. Figure 4 shows the experimental results for "static" networks. That is, the number $n(t)$ of nodes is fixed at 5,000 during the simulation. Random graphs with n nodes are generated as follows: each pair of nodes is connected with probability of $5.0/(n-1)$. Scale-free networks are generated using the incremental method proposed by Balabasi and Albert [17]. More precisely, starting with 3 fully connected nodes, we add new nodes one by one. When a new node is added, three links are also added to connect the node to three other nodes, which are randomly selected with probability proportional to their degrees.

Figure 4 shows transition of the number $r(t)$ of replicas with time t. In Fig. 4, the x-axis is a time scale and the y-axis is the total number of replicas in the network at the time. It shows the results for four combinations of two values of δ (0.02 and 0.01), and two initial number $r(0)$ of replicas (200 and the half of the target number). The length $CYCLE$ of the time interval is set to 1,000 time units. From Fig. 4, we can see that the number of replicas is decreased sharply during the early period in the case that the initial number of replicas is much larger than the target number. The reason of this is that most replicas can eat a small amount of food but the amount is not enough to survive. The number of replicas converges to the equilibrium point after a configuration where some replicas can eat the amount RF of food. These simulation results show that the number of replicas has small perturbation after the convergence. In addition, the average of the difference ratios, which is calculated by the sum of difference ratios at time $t(0 \le t \le 30,000)$ divided by the runtime, is about 0.06 after convergence of the replica density.

Simulation results for dynamic networks. Figure 5 shows the results for "dynamic" networks where some nodes join in the network and some nodes leave from the network constantly. When a new node joins in the network, the new node is connected to other nodes with probability $5.0/n(t)$ for each other node on random networks, and the new node is connected to three other nodes randomly selected with probability proportional to their degrees on scale-free networks. On joining in the network, the new node creates a new agent and sets the local clock to 0. When a node v leaves from the network, the links connecting to v, and replicas and agents on v or these links are also removed from the network. In the simulations, each node leaves from the network with a constant probability for random networks. For scale-free networks, the leave of a hub node having tremendous number of connections is undesirable for us since a lot of partitions will be caused. Therefore, for scale-free networks, a node v leaves from the network with probability $avg\_deg \cdot p_l/deg_v (0 < p_l < 1)$, where avg_deg is the average degree of the network. Notice that we keep the networks connected in the simulations: if a network is partitioned into some connected components due to leave of a node, we add a link randomly between two components.

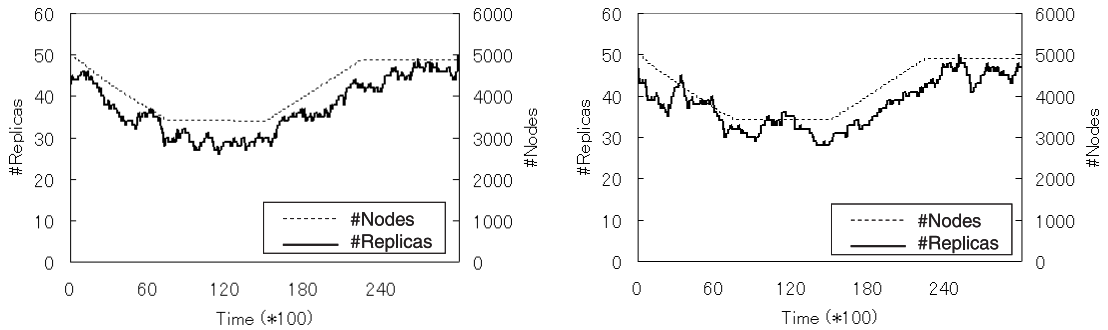
In this simulation, the initial network size $n(0)$ is 5,000, and the following dynamical changes occur every 200 time units. In the first quarter term (from time 0 to 7,500) of the simulation, a single new node joins with probability 0.1 and each node leaves with probability 0.01 for random networks and with probability $avg\_deg \cdot 0.01/deg_v$ for scale-free networks, that is, the network size is decreasing in this term. In the second quarter term (from time 7,500 to 15,000) and the fourth quarter term (from time 22,500 to 30,000), a single new node joins with probability 0.2 and each node leaves with probability 0.0001 for random networks and with probability $avg\_deg \cdot 0.0001/deg_v$ for scale-free networks, that is, the network size stays about the same in these terms. In the third quarter term (from time 15,00 to 30,000), 40 nodes joins with probability 1.0 and no node leaves from the network, that is, the network size is increasing in this term.

Figure 5 shows transition of the number $r(t)$ of replicas



a. Random networks ($n(t) = 5000$) b. Scale-free networks ($n(t) = 5000$)

Fig. 4 Simulation results of the first algorithm on static networks.



a. Random networks ($n(0) = 5000, r(0) = 50, \delta = 0.01$) b. Scale-free networks ($n(0) = 5000, r(0) = 50, \delta = 0.01$)

Fig. 5 Simulation results of the first algorithm on dynamic networks.

and the network size $n(t)$ with time t. The x-axis in the figure is a time scale and the y-axes on the left and the right are the total number of replicas and nodes in the network at the time respectively. The vertical scale of the network size is marked so that the transition of the network size corresponds to the target number. In the simulation results of Fig. 5, the length *CYCLE* is set to 1,000 time units, the value of δ is set to 0.01 and the initial number $r(0)$ of replicas is set to 50. The simulation results show that the number of replicas is adaptively adjusted in response to changes in the network size. In addition, the average difference ratio is 0.11. The total number of replicas is smaller than the target number for most of simulation time. The cause is the deletions of replicas and agents on nodes that leave from the networks and the delay until convergence of the replica density. Due to a leave of a node, replicas and agents with food on the node are also deleted. Since the amount of food supplied to replicas is decreased, the number of replicas in the network becomes smaller than the target number. When the network size is increasing, the amount of supplied food is also increasing, but new replicas cannot be created immediately. It takes a time to converge, but the number of replicas converges to the target number after a certain time as shown in Fig. 4.

Simulation results on lifetime of replicas. The goal of the replica density control problem is to adjust the number of replicas to a given ratio of the network size. A straightforward algorithm to achieve the goal is one based on probability: at a time interval, each node deletes all replicas it has and creates a replica with a probability δ. This naive algorithm has high adaptability, however, it causes frequent creations and deletions of replicas. From the point of application view, a frequency of creations and deletions of replicas should be low since copy and allocation cost of a replica is not ignored. Moreover, in terms of searching performance, locations of each replica should not be changed frequently. In real applications, if there is almost no change of locations of each replica, the searching performance can be improved. On the other hand, algorithms that do not create and delete replicas have low adaptability. Thus, we show our algorithm achieves a balance between adaptability and stability by showing that lifetime of replicas is sufficiently long.

Lifetime lt_x of replica x is defined to be the time length from its creation to its deletion, i.e., $lt_x = td_x - tc_x$, where td_x is the time when x is deleted and tc_x is the time when x is created. Table 1 shows the average lifetime of replicas of ten trials. To focus on the lifetime of replicas after convergence of the number of replicas to the target number, we count lifetimes of only replicas that are deleted the last half of the runtime. In the simulations, the value of δ is set to 0.01. The simulation results show that lifetime quickly becomes longer when the length of the time interval *CYCLE* becomes longer. The reason of this is that each replica with long *CYCLE* can meet more agents in *CYCLE* time units than with

Table 1 Average lifetime of replicas.

a. Random networks

		CYCLE		
		500	1000	1500
n	2000	1447	7825	16123
	5000	1398	7740	17012
	10000	1393	8286	18084

b. Scale-free networks

		CYCLE		
		500	1000	1500
n	2000	2275	5545	10267
	5000	2143	6202	12830
	10000	2043	6817	12317

short *CYCLE*. That is, replicas with long *CYCLE* are more likely to survive by being supplied food from many agents. Therefore, by setting an appropriate value to *CYCLE*, it is strongly expected that lifetime of each replica becomes sufficiently long.

Replica density algorithm by using smaller number of agents. In the algorithm presented in Sect. 4.1, $n(t)$ agents are created and traverse the network every *CYCLE* time units. When the number of replicas stabilizes to the target number, most of agents should feed all food for *CYCLE* time units since the average lifetime of replicas is long. In the simulation setting, it is expected that $n(t)/CYCLE$ nodes create agents at a time t. Thus, there are about $n(t)$ agents traversing the network at any time.

Although the size of the agent is so small since the agent has only information of food, the number $n(t)$ of agents may be large for the system. To reduce network traffic, we can modify the algorithm so that each node can create an agent every $c{\cdot}CYCLE$ time units for some constant c ($c > 1$). This method reduces network traffic of agents by $1/c$. Since the number of agents on the whole network is roughly reduced to $1/c \cdot n(t)$, each agent has to be created with the initial amount of food $c \cdot \delta \cdot RF$ to keep the total amount of food to be supplied. In this regard, however, the length of the time interval *CYCLE* needs to become longer depending on the value of c. The reason is that agents with larger amount of food must visit more nodes to supply food to more replicas.

We have verified by simulations that the network traffic of agents can be reduced to $1/c \cdot n(t)$ without sacrificing accuracy of the replica density control.

5. Algorithm for Replica Densities Control of Multiple Resources

For the objective of controlling multiple resources, the straightforward algorithm is to apply independently the algorithm presented in Sect. 4.1 to each resource. In this case, the number of agents on the network is also proportional to the number of resources. Moreover, each node needs the exact knowledge about all resources on the network, since it must create agents for each resource.

The goal of the algorithm presented in this section is to adapt the total number of replicas of all resources to $\delta \cdot n(t)$ and the number of replicas of each resource equally. This algorithm is based on the algorithm in Sect. 4.1. In this algorithm, however, only $n(t)$ agents are created every *CYCLE* time units regardless of the number of resources. Each node needs to know only the rough number (instead of the exact number) of resources.

5.1 Algorithm

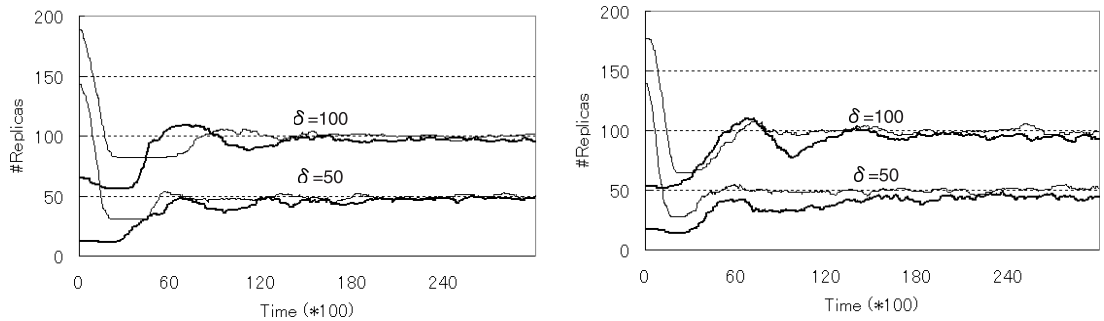
We consider how each agent should supply food to replicas. To adjust the total number of replicas to $\delta \cdot n(t)$, each agent should supply food of amount $\delta \cdot RF$ as the first algorithm. However, since the initial replica density of each resource may be different from that of every other resource, each replica density cannot be adjusted appropriately by the same way as the first algorithm. That is, the replicas in larger number can be fed more food than the replicas in smaller number, and keep the larger number.

To balance the replica densities of all resources, we introduce an additional condition for food supply: each agent tries to supply a same amount of food to replicas of each resource. However, each agent cannot know the number of resources on the network. Therefore, in the algorithm, we set an upper limit of the amount of food each agent can supply to a replica based on the amount of food the agent has supplied. Each agent p has information $sf_p(i)$ about the amount of food the agent p supplies to replicas of each resource i. Let SF_p be the maximum amount of food agent p has supplied to replicas of resources, that is, $SF_p = \max\{sf_p(i)\}$. At the agent p arriving at a node v, the agent p can supply the replica of resource i the following amount $F_p(i)$ of food.

$$F_p(i) = \begin{cases} food_p & \textbf{if } \forall j\, SF_p = sf_p(j) \\ SF_p - sf_p(i) & \text{Otherwise} \end{cases}$$

The expression $F_p(i)$ represents that the upper limit of the amount of food an agent supplies is the maximum amount of food the agent has supplied to replicas of a resource. That is, in the case that the maximum amount of food an agent p has supplied is x to replicas of a resource i, p will supply less than amount x of food to a replica of other resources while p will supply no food to a replica of the resource i. When p has supplied the same amount of food to replicas of all resources p knows, p will supply the amount of food p has to a next replica.

In this way of food supply, it is important for each agent to supply food to replicas of many resources. However, when the value δ is much smaller than the number of resources, each agent can supply food to only replicas of δ resources before it kills itself. In this case, the replicas that exist in large number can keep its large number since many agents can meet the replicas. Therefore, the amount of food each agent can supply at once should be limited to a little amount. The more resources exist in the network, the less amount of food each agent should supply to a replica to guarantee that the agent supplies food to replicas of many



a. Random networks ($n(t)$ = 5000) b. Scale-free networks ($n(t)$ = 5000)

Fig. 6 Simulation results of the second algorithm on static networks.

resources. On the other hand, the larger amount of food each agent has, the larger amount of food each agent should supply to a replica since each agent has to supply the amount $\delta \cdot RF$ of food during *CYCLE* time units. Thus, in the proposed algorithm, the amount of food each agent supplies to a replica on a visited node is less than $RF \cdot \delta/(C \cdot kind(t))$, where $kind(t)$ is the number of resources at time t and C is a positive value.

Based on the above discussion, the amount of food each agent p supplies to a replica i_h of resource i on a visited node is represented by

$$\min\{RF - eat\_food(i_h), food_p, F_p(i), RF \cdot \delta/(C \cdot kind(t))\}.$$

In the above method, it is necessary for each node to know the number $kind(t)$ of resources at time t. To estimate the number $kind(t)$, each node v has the set rsc_list_v of resource identifiers believed to exist in the network. The node v can obtain the information about existing resources from the agents visiting v, that is, the identifiers of resources the visiting agent has fed food are added to the set rsc_list_v.

Another problem we should consider is deletion of resources from the network. We consider dynamic networks that resources may be deleted by their original node. When an original node of a resource i decides deletion of i, the replicas of resource i also need to be deleted from the network. To feed no food to the replicas of the deleted resource, an original node keeps the set of identifiers of resources the node has deleted. Each agent p has the set del_list_p and collects the identifiers of the deleted resources on the visited nodes. If agent p finds the replica of the resource included in the set del_list_p, the replica is deleted from the network.

5.2 Simulation Results

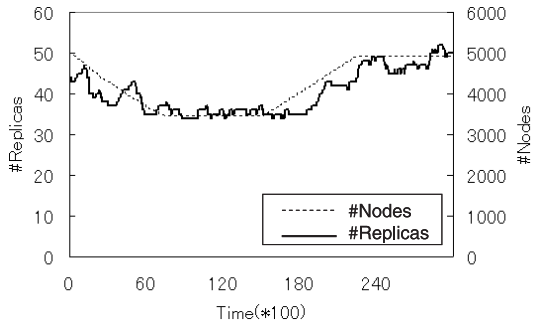
Now, we show simulation results of the second algorithm. In the simulation, the behavior and the initial locations of agents are determined in the same way as that in Sect. 4.2. The positive value C is set to 1.0.

Simulation results for static networks. Figure 6 shows the results for "static" networks. The network is created by the way presented in Sect. 4.2. In the simulation, the length *CYCLE* of the time interval is set to 1,000. The network size $n(t)$ is fixed at 5,000 during the simulation. The number of resources is set to 5,000, and the value δ is set to 50 and 100. That is, the number of replicas of each resource should be kept to 50 and 100 respectively. The initial number of replicas of each resource is set to a random value from 1 to 200. In Fig. 6, we show the transition of the replica density of two resources chosen randomly from the 5,000 resources on each case of δ is 50 and 100. The x-axis is a time scale and the y-axis is the number of replicas of the chosen resources in the network at the time in Fig. 6.

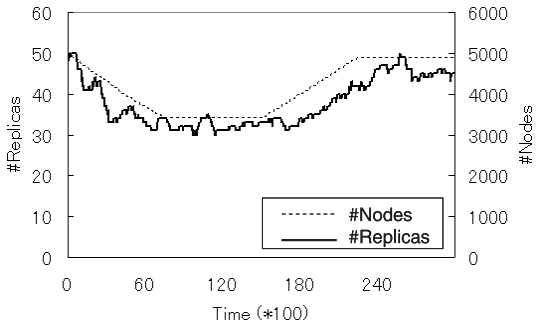
The simulation results in Fig. 6 show that the number of replicas converges to and stabilizes at the target number respectively. The average difference ratio of 5,000 resources at the end of the simulation is less than 0.07. In the case that the initial number of replicas is large, the number of replicas is not decreased as much as Fig. 4. This is caused by difference of the amount of food an agent supplied. In this simulation, each replica can eat enough amount of food by meeting only one agent since a created agent initially has $50 \cdot RF$ amount of food while each replica must meet many agents to survive in the simulation of Fig. 4.

Simulation results for dynamic networks. Figure 7 shows the simulation results for "dynamic" networks in which the network size $n(t)$ varies with time. The networks change in the same way as the simulation in Sect. 4.2. Note that the number of resources remains unchanged: when the original node v of resource i leaves from the network, node v sends the original resource i to one of its neighbors, and the neighbor becomes the original node of the resource i. In the simulation, the length *CYCLE* of the time interval is set to 1,000, the number of resources is set to 5,000, and the value δ is set to 50. Since the network size varies with time, the target number of replicas of each resource also varies with time. The target number of replicas of each resource is 50 at time 0, about 35 at time 7,500 and about 48 at the end of the simulation. The initial number of replicas of each resource is set to 50.

In Fig. 7, we show the transition of the number of replicas of a resource chosen randomly from 5,000 resources. The y-axes on the left and the right are the number of repli-

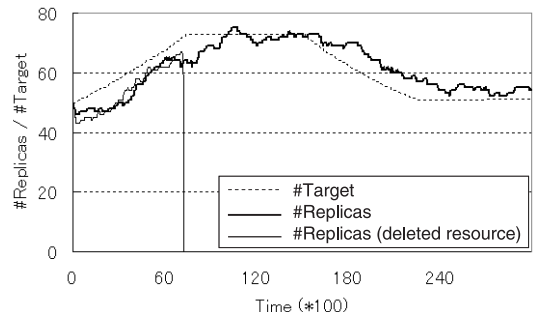


a. Random networks ($n(0) = 5000, r(0) = 50, \delta = 50$)

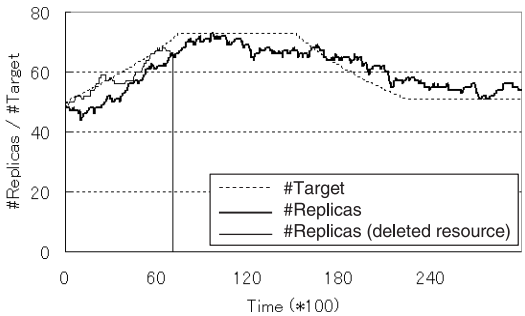


b. Scale-free networks ($n(0) = 5000, r(0) = 50, \delta = 50$)

Fig. 7 Simulation results of the second algorithm on dynamic networks with changes of the network size.



a. Random networks ($n(0) = 5000, r(0) = 50, \delta = 50$)



b. Scale-free networks ($n(0) = 5000, r(0) = 50, \delta = 50$)

Fig. 8 Simulation results of the second algorithm on dynamic networks with changes of the number of resources.

cas of the chosen resource and nodes in the network at the time respectively. These simulation results also show the replica density is adaptively adjusted in response to changes in the network size.

In the simulations presented on Fig. 8, the number of resources changes. When the number of resources becomes larger, the target number of replicas of each resource becomes smaller. On the other hand, when the number of resources becomes smaller, the target number becomes larger. In the simulation results of Fig. 8, the network size $n(t)$ is fixed at 5,000, the length *CYCLE* is set to 1,000 time units, the value of δ is set to 50 and the initial number of replicas of each resource is set to 50. The initial number of resources is 5,000 (the target number is 50), and about 1,550 resources are deleted from the network at time 10,000 of the simulation (the target number becomes about 73), and about 1,550 resources are added to the network at time 20,000 of the simulation (target number becomes about 50). The deleted resources are chosen randomly.

In Fig. 8, the y-axis is the number of replicas of the chosen resources in the network at the time. The simulation results show the adaptation of the replica density to changes in the number of resources. In addition, these results also show that replicas of deleted resources quickly deleted from the network because of no supplied food. When new resources are adding in the networks, the target number of replicas of each resource is decreasing. In the simulation, however, the number of replicas of a preexisting resource cannot be decreased immediately. The reason of this is that immediately after a new resource is added, the number of replica of the new resource is small and the replicas of the preexisting resource are supplied food for the new resource.

6. Conclusions

In this paper, we have proposed two distributed algorithms for the replica density control problem. The problem requires us to adapt the total number of replicas to a given constant fraction of the current network size and adapt the number of replicas of each resource equally. The algorithms are inspired by the single species population model. The first algorithm is very simple and easily understandable. The algorithm can adapt the current density of a single resource to the current network size. To control multiple resources by using the first algorithm, a large number of agents and the identifiers of every resource are required at each node. The second algorithm controls densities of all resources. As a result, the number of agents created periodically is $n(t)$ regardless of the number of resources. Moreover, in the second algorithm, each node estimates the number of resources to supply food at an amount corresponding to the number of resources, but does not need to have the exact knowledge about all resources. The simulation results show that the proposed algorithms can adequately adjust the number of

replicas in dynamic networks. In addition, from the simulation results, the lifetime of each replica becomes sufficiently long by setting an appropriate value to algorithm parameter *CYCLE*.

By applying our algorithms, it is also possible to control the number of replicas of each resource depending on its popularity if a popularity distribution of resources is given. One of the methods to realize popularity-based control is that some copies of an original resource are created depending on its popularity and identified as different resources (e.g., each copy is assigned a different resource ID). Since our algorithm keeps the number of replicas of each resource equally, the method can keep the number of replicas of a resource proportional to its popularity.

In this paper, we focus on only the number of replicas. In real systems that provide resource replication, allocation of replicas is also very important. Our future work is to develop the replica allocation algorithm for improving system performance: agents determine allocations of new replicas from network conditions that agents can learn by traversing over the network. Another future work is to propose a migration pattern of an agent for supplying food efficiently. In this paper, each agent makes a random walk independently. It remains possible that agents can supply food efficiently by using higher technical migration pattern such as a walk based on the amounts of food neighbor nodes have or the degree of neighbor nodes.

Acknowledgements

This work is supported in part by Grant-in-Aid for Scientific Research((B)19300017, (B)17300020, (B)20300012) of JSPS, Grant-in-Aid for Young Scientists ((B)18700059, (B)19700058) of JSPS, Global COE (Centers of Excellence) Program of MEXT, the Hori Information Science Promotion Foundation, and the Kayamori Foundation of Informational Science Advancement.

References

[1] "The bio-networking architecture," http://netresearch.ics.uci.edu/bionet/
[2] "The anthill project," http://www.cs.unibo.it/projects/anthill/
[3] O. Babaoglu, H. Meling, and A. Montresor, "Anthill: A framework for the development of agent-based peer-to-peer systems," Proc. 22th International Conference on Distributed Computing Systems, pp.15–22, 2002.
[4] J. Suzuki and T. Suda, "Design and implementation of a scalable infrastructure for autonomous adaptive agents," Proc. 15th IASTED International Conference on Parallel and Distributed Computing and Systems, pp.594–603, Nov. 2003.
[5] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," Proc. Workshop on Design Issues in Anonymity and Unobservability, pp.46–66, July 2000.
[6] Gnutella.com. http://www.gnutella.com
[7] K. Miura, T. Tagawa, and H. Kakugawa, "A quorum-based protocol for searching objects in peer-to-peer networks," IEEE Trans. Parallel Distrib. Syst., vol.17, no.1, pp.25–37, 2006.
[8] Y. Drougas and V. Kalogeraki, "A fair resource allocation algorithm for peer-to-peer overlays," Proc. 24th Conference on Computer Communications, pp.2853–2858, March 2005.
[9] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, pp.586–595, Jan. 1999.
[10] S. Tewari and L. Kleinrock, "Proportional replication in peer-to-peer networks," Proc. 25th IEEE International Conference on Computer Communications, April 2006.
[11] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," Proc. 2002 SIGCOMM Conference, pp.177–190, Oct. 2002.
[12] T. Suzuki, T. Izumi, F. Ooshita, and T. Masuzawa, "Self-adaptive mobile agent population control in dynamic networks based on the single species population model," IEICE Trans. Inf. & Syst., vol.E90-D, no.1, pp.314–324, Jan. 2007.
[13] T. Suzuki, T. Izumi, F. Ooshita, H. Kakugawa, and T. Masuzawa, "Bio-inspired replica density control in dynamic networks," Proc. 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology, Jan., 2006.
[14] V.A. Pham and A. Karmouch, "Mobile software agents: An overview," IEEE Commun. Mag., vol.36, no.7, pp.26–36, July 1998.
[15] A.R. Silva, A. Romao, D. Deugo, and M. Mira, "Towards a reference model for surveying mobile agent systems," Autonomous Agents and Multi-Agent System, vol.4, no.3, pp.187–231, 2001.
[16] R. Haberman, Mathematical Model: Population Dynamics, Prentice Hall, 1977.
[17] R. Albert and A.L. Barabasi, "Statistical mechanics of complex networks," Reviews of Modern Physics, vol.74, no.1, pp.47–97, Jan. 2002.
[18] A.L. Barabasi and E. Bonabeau, "Scale-free networks," Scientific American, vol.288, pp.50–59, May 2003.

Tomoko Izumi received the B.E., M.E. and D.I. degrees in computer science from Osaka University in 2003, 2005 and 2008. She is now a postdoctoral researcher of Center for Social Contribution and Collaboration, Nagoya Institute of Technology. Her research interests include distributed algorithms. She is a member of IEEE.

Taisuke Izumi received the M.E. and D.I. degrees in computer science from Osaka University in 2003 and 2006. He is now an Assistant Professor of Graduate School of Engineering, Nagoya Institute of Technology. His research interests include distributed algorithms. He is a member of ACM and IEEE.

Fukuhito Ooshita received the M.E. and D.I. degrees in computer science from Osaka University in 2002 and 2006. Since 2003, he has been an Assistant Professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include parallel algorithms and distributed algorithms. He is a member of ACM, IEEE, and IPSJ.

Hirotsugu Kakugawa received the B.E. degree in engineering in 1990 from Yamaguchi University, and the M.E. and D.E. degrees in information engineering in 1992, 1995 respectively from Hiroshima University. He is currently an associate professor of Osaka University. His research interests include distributed algorithms. He is a member of IEEE Computer Society and Information Processing Society of Japan.

Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Osaka University during 1987–1994, and was an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) during 1994–2000. He is now a professor of Graduate School of Information Science and Technology, Osaka University. He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, EATCS and the Information Processing Society of Japan.