Self-Organizing Network Services With Evolutionary Adaptation

Tadashi Nakano, *Member, IEEE,* and Tatsuya Suda, *Fellow, IEEE*

***Abstract*—This paper proposes a novel framework for developing adaptive and scalable network services. In the proposed framework, a network service is implemented as a group of autonomous agents that interact in the network environment. Agents in the proposed framework are autonomous and capable of simple behaviors (e.g., replication, migration, and death). In this paper, an evolutionary adaptation mechanism is designed using genetic algorithms (GAs) for agents to evolve their behaviors and improve their fitness values (e.g., response time to a service request) to the environment. The proposed framework is evaluated through simulations, and the simulation results demonstrate the ability of autonomous agents to adapt to the network environment. The proposed framework may be suitable for disseminating network services in dynamic and large-scale networks where a large number of data and services need to be replicated, moved, and deleted in a decentralized manner.**

***Index Terms*—Adaptive and scalable network services, autonomous agents, evolutionary computation, self-organization, swarm intelligence.**

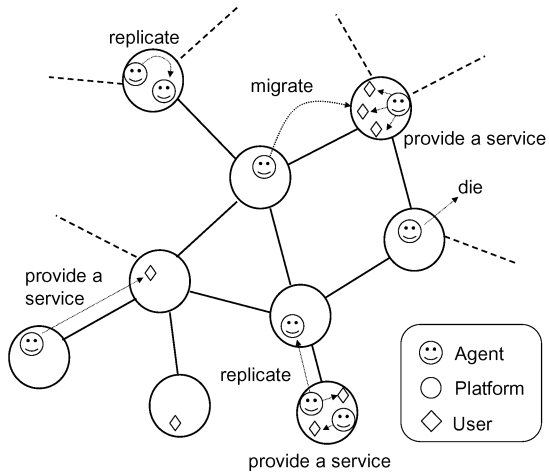


Fig. 1. Network service provided by a group of autonomous agents.

I. Introduction

A KEY challenge in communication networks is to design network services that adapt to dynamically changing network conditions and scale to the increasing number of network nodes and users. One promising approach in designing such adaptive and scalable network services is to apply self-organization without centralized control. The dynamic nature and large scale of the network environments prohibit approaches that require centralized control.

This paper presents a framework for developing adaptive and scalable network services using a group of distributed and autonomous (or self-organizing) agents. Fig. 1 illustrates the framework proposed and investigated in this paper. In this framework, agents autonomously replicate, migrate, and die based on locally available information (e.g., resource availability of nearby nodes or platforms). Agents interact with each other and collectively provide a service to users. Agents use platform resources such as processing power, memory, and storage space in order to provide services.

Manuscript received November 7, 2003; revised April 23, 2005. This work was supported in part by the National Science Foundation under Grants ANI-0508506, ANI-0083074, and ANI-9903427, in part by the Defense Advanced Research Project Agency under Grant MDA972-99-1-0007, in part by the Air Force Office of Scientific Research under Grant MURI F49620-00-1-0330, and in part by grants from the California MICRO and CoRe Programs, Hitachi, Hitachi America, Novell, Nippon Telegraph and Telephone (NTT), NTT Docomo, Fujitsu, NS Solutions Corporation, DENSO IT Laboratory, and NICT.

The authors are with Donald Bren School of Information and Computer Sciences, University of California, Irvine, CA 92697 USA (e-mail: tnakano@ics.uci.edu; suda@ics.uci.edu).

Digital Object Identifier 10.1109/TNN.2005.853421

An important component of the framework proposed in this paper is an evolutionary adaptation mechanism that allows agents to evolve their behavior. Traditional evolutionary computation [18] is not applicable to distributed environments such as that assumed in this paper, since it relies on centrally-controlled selection of members of the population. The evolutionary adaptation mechanism described in this paper is designed for distributed environments and allows distributed agents to autonomously evaluate nearby agents (i.e., agents that lie on the platforms within n platform hops) and select from them. Through the evolutionary adaptation mechanism designed in this paper, distributed agents autonomously adapt their behavior to the network environment and improve their fitness values (e.g., response time to a service request) to the environment.

Autonomous agents in the framework proposed in this paper are abstract entities that represent network services. The framework may be used to create a variety of services including peer-to-peer file sharing services [21] and content distribution services [15], [17]. For instance, early work that the authors of this paper conducted on the proposed framework described a design of a content distribution service using the framework [29]. In the content distribution service designed using the proposed framework, an agent represents an HTTP server, and it replicates and migrates over the Internet based on user demand.

The framework proposed in this paper is based on distributed and autonomous agents interacting and collectively providing services. A similar concept is found in swarm intelligence [1], where simple agents collectively solve a problem in a flexible and decentralized manner. Swarm intelligence has been applied

to various fields including combinational and numerical optimization [3], [12], clustering [4], load-balancing in communication networks [2], [25] and robotics [14]. The framework described in this paper focuses on creation of services using interacting agents, and the evolutionary mechanism that agents use in the framework is significantly different from those developed in swarm intelligence.

As described earlier, a content distribution service may be implemented using the framework proposed in this paper. Although the framework proposed in this paper and existing frameworks to provide content distribution services [5], [7], [16], [22], [23], [24] share the goal of providing scalable network services by placing objects or services over distributed network nodes, the framework proposed in this paper fundamentally differs in how to achieve the goal. In the existing frameworks, object placement algorithms are designed (by human designers), assuming certain network characteristics. Thus, the existing frameworks inherently limit their capability to adapt to dynamically changing network conditions, especially when the dynamic changes are not considered at the time of designing object placement algorithms. On the other hand, the framework proposed in this paper allows agents (e.g., network services) to evolve and adapt their behaviors to changing network environments without human intervention.

The rest of this paper is organized as follows. Section II describes key features of the agents in the framework proposed in this paper. It also describes a design of agents and an evolutionary adaptation mechanism for the agents. Section III demonstrates the behavior of the agents through simulations. Section IV concludes the paper.

II. Framework for Self-Organizing Network Services

This section describes the autonomous agents in the framework proposed in this paper. It also presents an evolutionary adaptation mechanism for the agents.

A. Autonomous Agents: Overview

In the framework proposed in this paper, an agent consists of three parts: attributes, body, and behaviors. *Attributes* carry information regarding the agent (e.g., description of a service it provides and its age). The *body* implements a service that an agent provides and contains materials relevant to the service (e.g., data and application code). For instance, an agent may implement control software for a device in its body, while another agent may implement a hotel reservation service in its body. An agent that implements a web service may contain a web page data in its body. Agent *behaviors* implement nonservice related actions that are inherent to all agents. Examples of behaviors include migration, replication, reproduction (i.e., sexual replication), and death.

Each node in the network runs the platform software (referred to as the platform in the rest of the paper), which provides an execution environment and supporting facilities for agents. The platform software also manages underlying resources such as CPU and memory. Agents run atop the platform software. The minimum requirement for a network node to participate in the framework proposed in this paper is to run the platform software.

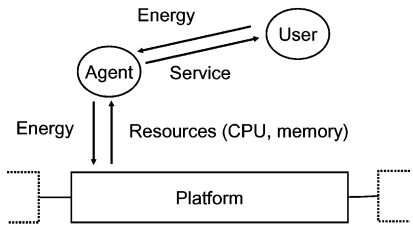


Fig. 2. Energy exchange.

Each agent in the framework proposed in this paper may store and expend energy for living. Agents may gain energy in exchange for performing a service, and they may pay energy to use platform resources (see Fig. 2). For instance, an agent provides a service to users, and in return, it may receive energy from users. An agent uses platform resources (i.e., CPU and memory) to invoke its behavior and to provide a service to users. An agent may then pay energy to the platform for using its resources.

The abundance or scarcity of stored energy in an agent may affect various behaviors and contributes to the natural selection process in evolution. For example, an abundance of stored energy is an indication of higher demand for the agent. Thus, the agent may be designed to favor reproduction in response to higher levels of stored energy. A scarcity of stored energy (an indication of lack of demand or ineffective behaviors) may eventually cause the agent to die.

B. Autonomous Agents: Behavior

Agents in the framework proposed in this paper are relatively simple and autonomous and follow simple behavior rules. Agents autonomously invoke their behavior based on their internal state (e.g., those shown in Table I) and environmental conditions that they sense in their local environment (e.g., those shown in Table II). Some of the agent behaviors are explained in the following.

- *Migration*. Agents may migrate from a platform to another platform. For instance, agents may migrate toward a user requesting their service when they are away from the user (i.e., when the hop count to the user is high). This may decrease user-perceived latency in receiving a service. Agents may also migrate to platforms with more reasonable resource costs. This may allow agents to save energy. Agents may also migrate away from platforms that are highly populated with agents. This may increase survivability of agents in case of platform failure.
- *Replication and reproduction*. Agents may make a copy of themselves (replication), possibly with mutation in the replica's behavior. Two parent agents may reproduce and create a child agent (reproduction), possibly with mutation and crossover in the child's behavior. For instance, agents may replicate or reproduce when their energy level is high. This allows better response to increased demand for the service since a high energy level indicates that the service provided by the agents is in high demand. Agents may also replicate or reproduce to populate a platform when the platform hosts a very small number of agents.

TABLE I
EXAMPLES OF AGENT INTERNAL STATE

Internal state	Definition
Age	The time elapsed since the birth of the agent
Energy level	The total amount of energy that the agent possesses
Number of services provided	The total number of users that the agent has provided a service since its birth
Waiting time	The interval between the time when a user requests a service and the time when the user starts receiving a service
Hop count	The number of platform hops that a service request traveled from a user to the agent
Activeness	The factor that influences the rate of behavior activity

TABLE II
EXAMPLES OF ENVIRONMENTAL CONDITIONS

Environmental condition	Definition
Request rate on a platform	The total number of service requests that the agents on the platform receive per time unit
Resource cost of a platform	Energy cost of using resources on the platform
Agent population size on a platform	The number of agents on the platform
Request rate change on a platform	Change of the request rate on the platform
Resource availability of a platform	Available resources (e.g., CPU and memory) of the platform
Behavior invocation cost of a platform	Energy cost of performing behavior on the platform

This may increase the availability of their service and also prevent possible extinction of agents.

- *Death*. Agents may die because of lack of energy. If energy expenditure of an agent is not balanced by the energy units it receives, it will not be able to pay for the platform resources it needs, i.e., it dies from lack of energy. Agents with wasteful behaviors (e.g., replicating or migrating too often) will have a higher chance of dying from lack of energy. Agents may also die when the agent population on a platform is large. This may prevent the overuse of platform resources.

C. Autonomous Agents: Behavior Invocation Algorithms

An agent invokes its behavior (such as replication, reproduction, migration, and death) based on its internal state (Table I) and environmental conditions (Table II). Each agent behavior may be implemented using the weighted sum aggregation method proposed in [19]. Fig. 3 illustrates the weighted sum aggregation method. In this method, an agent carries a set of factors (denoted as v_i in Fig. 3), a weight associated with a factor (denoted as w_{xi} in Fig. 3) and a threshold (denoted as θ_x in Fig. 3) that govern its behavior. Factors are agent's internal state and environmental conditions. An agent invokes behavior x when $\Sigma v_i \cdot w_{xi} > \theta_x$, where v_i is a factor, w_{xi} is a weight associated with the factor v_i, and θ_x is the threshold for behavior x. The following describes a specific implementation of four behaviors (migration, replication, reproduction, and death) that an agent may have using the weighted sum aggregation method. The implemented algorithms are later examined through simulations in Section III.

1) Migration: In migration, an agent first examines each adjacent platform and obtains its weighted sum $(\Sigma v_i \cdot w_{\mathrm{mig}\_i})$.

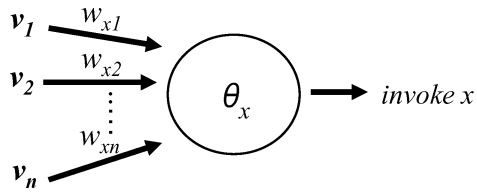


Fig. 3. Weighted sum aggregation method for behavior invocation.

It also obtains the weighted sum for the platform that it currently resides. The agent migrates toward the platform with the highest sum if the highest sum exceeds its migration threshold (θ_{mig}). Otherwise (i.e., if the highest sum is smaller than the threshold, or if the highest sum is at the current platform), it stays at the current platform. For instance, consider an example illustrated in Fig. 4 under the environmental conditions shown in Table III, where two platforms (P_c and P_e) satisfy the equation $(\Sigma v_i \cdot w_{\mathrm{mig}\_i} > \theta_{\mathrm{mig}})$; and the sum $(15 \times 1.0 + 5 \times 0.5 = 17.5)$ at P_c is the higher than that $(10 \times 1.0 + 5 \times 0.5 = 12.5)$ at P_e. In this case, an agent migrates toward P_c, the platform with the highest sum.

2) Replication: An agent replicates when its weighted sum for replication $(\Sigma v_i \cdot w_{\mathrm{repl}\_i})$ exceeds its replication threshold (θ_{repl}). In replication, a newly replicated agent may derive its behaviors from its parent with mutation. In mutation, a new set of behaviors is derived from the parent's behaviors by changing weights in behavioral factors or removing/adding factors. (See mutation description in reproduction in the following.) A parent agent then provides a part of its energy to a child agent and places the child agent on the platform that the parent agent resides.

3) Reproduction: An agent reproduces when its weighted sum for reproduction $(\Sigma v_i \cdot w_{\mathrm{repr}_{\subset}})$ exceeds its reproduction

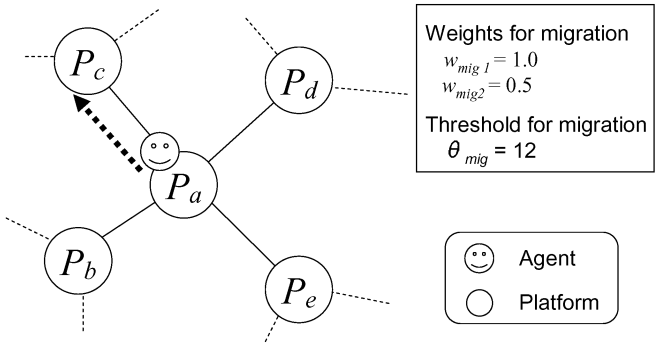


Fig. 4. Migration under the environmental conditions shown in Table III.

TABLE III
ENVIRONMENTAL CONDITIONS IN FIG. 4

Adjacent platform	v_1 (Request rate)	v_2 (Resource availability)	Weighted sum
P_a	5	5	7.5
P_b	5	10	10
P_c	15	5	17.5
P_d	0	10	5
P_e	10	5	12.5

threshold (θ_{repr}). In reproduction, an agent selects a reproduction partner from agents that are within n platform hops.

Agents within n platform hops are first evaluated based on their fitness. Three fitness measures are used: the average *waiting time* (i.e., the interval between the time when a user requests a service and the time when the user starts receiving a service), the average *hop count* (i.e., the number of platform hops that a service request travels from a user to the agent that provides the requested service), and the average *energy efficiency* (i.e., the fraction of the consumed energy and acquired energy). These fitness measures of an agent are averaged over time from the birth of the agent.

Among the three fitness measures described previously, an agent dynamically decides which measure to use in partner selection based on the preference-based multiobjective evolutionary optimization method [6]. Namely, given a set of preferred (or target) fitness measure values, the agent examines the level of satisfaction for each of the three fitness measures by calculating $ls_i = p_i/c_i$, where p_i is a given preferred fitness measure value and c_i is its current fitness measure value (i.e., fitness measure value of the agent who is to reproduce). The agent, then, uses the fitness measure k with a probability of $ls_k/\Sigma ls_i$. This allows an agent to use the fitness measure that needs most improvement. After selecting the fitness measure, the agent sorts the candidate agents and probabilistically selects a reproduction partner based on the rank of the candidate agents. The probability of an agent being selected linearly increases according to the rank of the agent as in the linear rank selection method commonly used in genetic algorithms (GAs) [18].

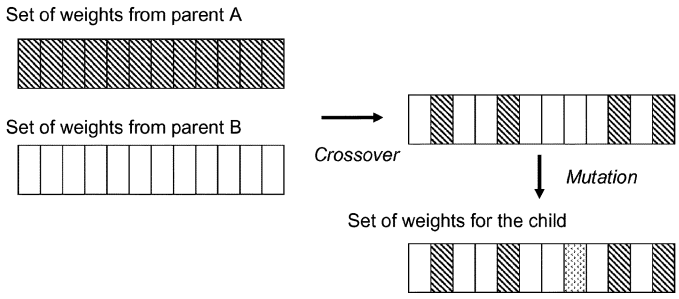


Fig. 5. Crossover and mutation.

After a reproduction partner is selected, an agent reproduces with the partner agent. In reproduction, crossover, and mutation of agent behaviors occur as shown in Fig. 5. Crossover mixes the behavioral weights of parents to produce new behavioral weights in offspring. In creating a set of weights for a child agent, more numbers of weights are inherited from the fitter parent. Specifically, the probability of a weight being chosen from a parent is in proportion to the number of the parent's fitness measures that are greater in value than those of the other parent.

After crossover, mutation may occur at each weight of a child with certain probability (referred to as the *mutation rate* in the rest of the paper). In mutation, each weight value is subject to a random change within a range of values (referred to as the *mutation range* in the rest of the paper).

4) Death: An agent dies when it exhausts energy.

Implementation of the behavior algorithms described in this section is based on the weighted sum aggregation method. Although the weighted sum algorithm method is simple, it creates wide varieties of agent's behaviors by simply increasing the number of factors and weights. Note that agent behaviors can also be implemented using other methods such as artificial neural networks [30] and genetic programming [13]. Note also that replication and reproduction may be extended by applying existing techniques in the evolutionary computation literature, including other crossover and mutation operators [18], multiobjective optimization [6] and self-adaptive parameter control [8].

III. SIMULATION STUDY

A simulation study is carried out to investigate the framework proposed in this paper. The following describes the simulation model and presents the simulation results.

A. Simulation Model

In the simulation, an agent implements migration, replication (without mutation), reproduction with crossover and mutation, and death behaviors. In order to simulate users, platforms generate service requests in the simulation. A service request is forwarded to an agent that is not busy (i.e., an agent that is not serving any service requests) and closest to the platform that generated the service request. If all agents are busy, a service request remains in the queue on the platform where it was generated until an agent becomes available. When an agent provides a service, the agent receives energy from the platform that generated the service request, while it expends energy at a constant rate for using platform resources.

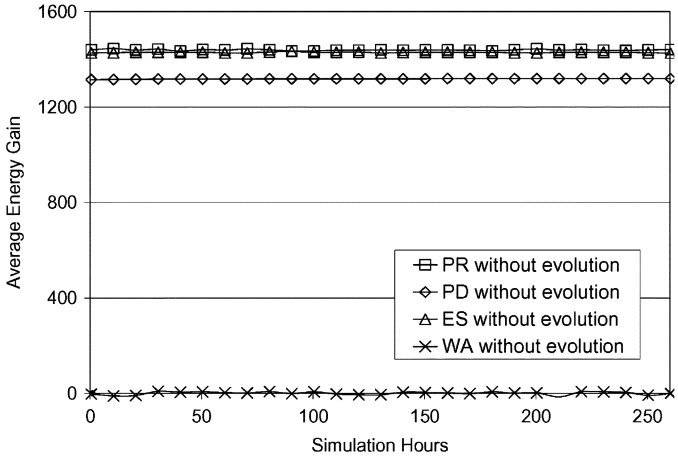


Fig. 6. Average energy gain without evolutionary adaptation.

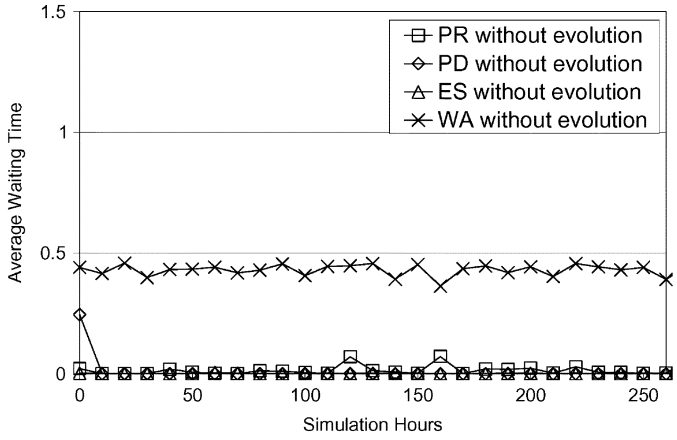


Fig. 7. Average waiting time without evolutionary adaptation.

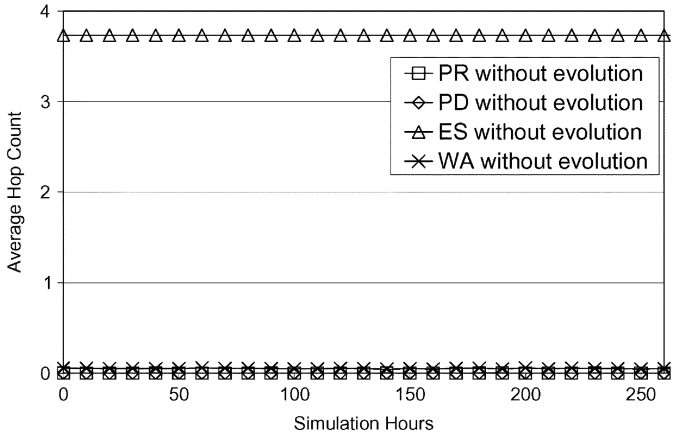


Fig. 8. Average hop count without evolutionary adaptation.

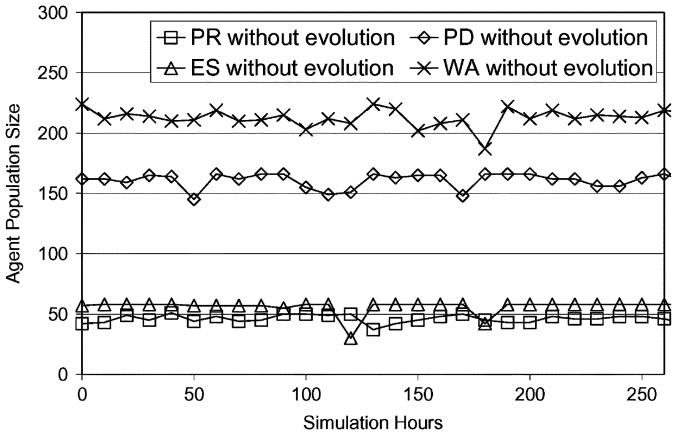


Fig. 9. Agent population size without evolutionary adaptation.

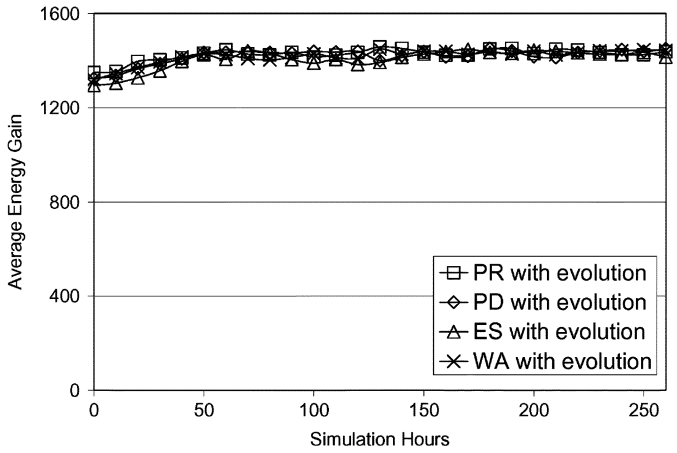


Fig. 10. Average energy gain with evolutionary adaptation.

In the simulation, the following performance measures are obtained to evaluate the evolutionary adaptation and the framework described in this paper: the average energy gain of agents (i.e., the difference between the total acquired energy and total consumed energy per second, averaged over all agents), the average waiting time of a service request (i.e., the average time interval between the time when a service request is generated at a platform and the time when the platform starts receiving the requested service), the average hop count of a service request (i.e., the average hop count that a service request traverses to reach the agent that provides the requested service), and the agent population size (i.e., the number of agents in the system).

B. Evolution in a Simple Network

The first set of simulations (Figs. 6–13) evaluates evolving agents in a relatively simple network environment. Table IV summarizes simulation parameter values used for the first set of simulations. The simulated network is that of an 8×8 mesh topology containing 64 platforms placed on each cross point. Seven platforms are randomly selected at the beginning of the simulation, and they generate service requests at a constant rate of 150 requests per second throughout the simulation.

The service time at an agent (i.e., time that an agent spends providing a service for a service request) is a constant, 0.2 s. An agent receives 10 energy units in return for a service. An agent expends 1 energy unit per second for using platform resources. An agent makes a decision on whether to replicate, reproduce, and migrate every 15 s, and expends 500 energy units in performing a behavior (i.e., replication, reproduction, and migration). An agent dies when it exhausts energy. In reproduction, an agent selects a reproduction partner either on the same platform or on the adjacent platforms. In both replication and reproduction, a parent agent (in case of replication) or the agent who decides to reproduce (in case of reproduction) provides 3000

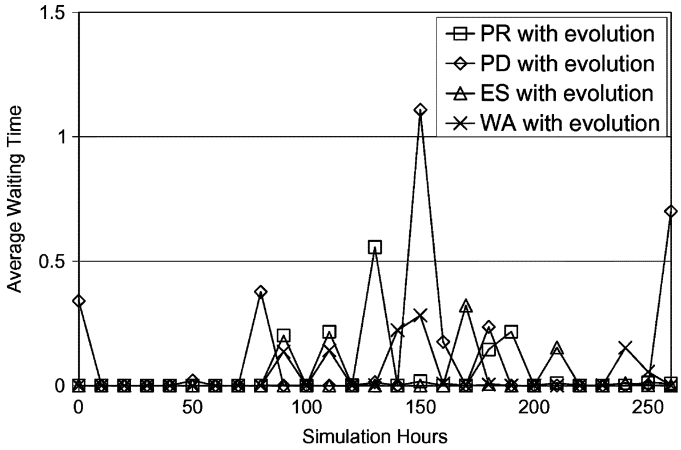


Fig. 11. Average waiting time with evolutionary adaptation.

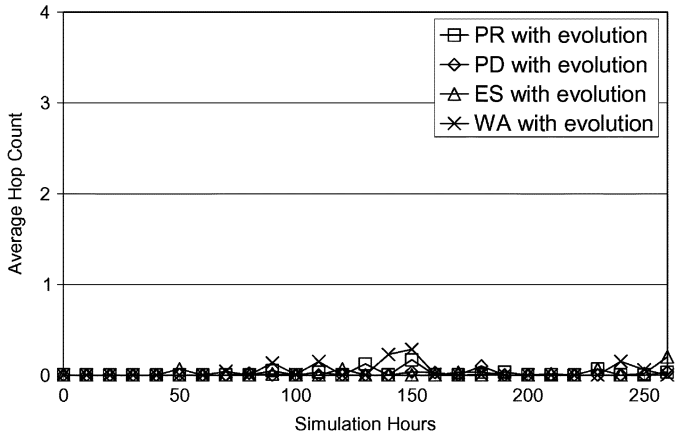


Fig. 12. Average hop count with evolutionary adaptation.

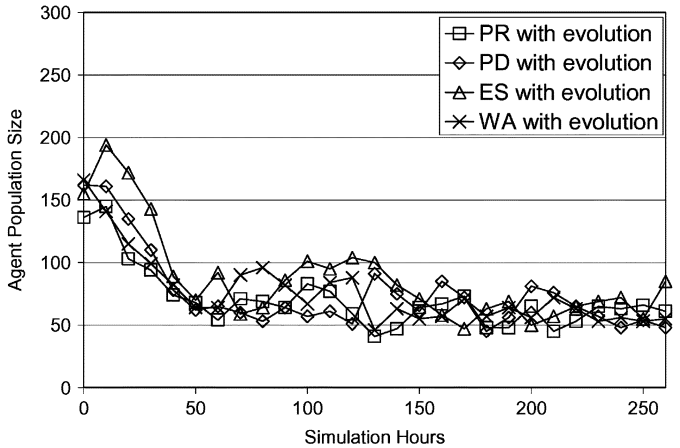


Fig. 13. Agent population size with evolutionary adaptation.

energy units to its child agent. In migration, an agent migrates only to an adjacent platform.

Each simulation in the first set of simulations (Figs. 6–13) was run assuming one of the following four types of agents, PR, PD, ES, WA. An agent of the primary (PR) type does not have any bias in its behavior. An agent of the productive (PD) type tends to replicate and reproduce more often than the other three types of agents. An agent of the energy seeker (ES) type tends to migrate to the adjacent platform that has the largest number of service requests in its queue and, thus, to stay at the platform once it migrates to such a platform. An agent of the wandering (WA) type randomly migrates among platforms. Table V lists a set of weights and thresholds for these four types of agents, where R represents weights associated with reproduction and replication, and M represents those with migration.

TABLE IV
DEFAULT SIMULATION PARAMETERS

Parameters	Values
the number of platforms	64
the generation rate of service requests	150 per second
Service time	0.2 seconds per service
Service cost	10 energy units per service
Platform resource cost	1 energy unit per second
Migration cost	500 energy units per migration
Reproduction cost	500 energy units per reproduction
Replication cost	500 energy units per replication
Interval to decide whether to invoke a behavior	15 seconds

1) Simulations Without Evolutionary Adaptation: In the simulations shown in Figs. 6–9, the evolutionary adaptation mechanism (i.e., reproduction with partner selection, crossover, and mutation) is tuned off, and the behavior weight values do not change throughout the simulations. Each simulation starts with only one agent with 10 000 energy units.

In Figs. 6–9, the horizontal axis indicates simulation time, while the vertical axis indicates performance measures described earlier (i.e., the average energy gain of agents in Fig. 8, the average waiting time of a service request in Fig. 6, the average hop count of a service request in Fig. 7, and the agent population size in Fig. 9).

Simulation results show the following; PR achieves the highest energy gain and, thus, is most energy efficient (Fig. 6); PD achieves small waiting times (Fig. 7), but it achieves lower energy gain compared to PR and ES due to its excessive replication (Fig. 6); ES shows the highest average hop count, and this is because, once it migrates to a platform with the largest number of service requests in the queue, it no longer migrates toward other platforms generating service requests (Fig. 8); WA exhausts energy mostly from migration, as this type of agents randomly migrates regardless of where service requests are generated (Fig. 6).

2) Simulations With Evolutionary Adaptation: In Figs. 10–13, the same set of simulations as those in Figs. 6–9 was run with the evolutionary adaptation mechanism (i.e., reproduction with partner selection, cross over, and mutation) turned on. In these simulations, the set of weights listed in Table V may undergo changes through crossover and mutation with the mutation rate of 0.2 and the mutation range of 0.1. In

TABLE V
FOUR AGENT TYPES USED IN SIMULATIONS

Weights (w_i)	PR	PD	ES	WA
R.request rate	0.1	0.3	0.1	0.1
R.threshold	0.5	0.5	0.5	0.5
M.request rate	1.0	1.0	2.0	0.2
M.resource cost	0.5	0.5	0.0	0.5
M.agent population size	1.0	1.0	0.5	0.3
M.activeness	0.5	0.5	0.5	1.0
M.threshold	2.0	2.0	2.0	2.0

the simulations, mutation is applied only to the weights and not to the thresholds. The preferred fitness measure values are 0.3 for the average energy gain, 1.0 s for the average waiting time, and 0.5 hops for the average hop count.

By comparing simulation results in Figs. 10–13 with those in Figs. 6–9, the following may be observed. PR does not evolve to improve any of the three measures (average energy gain, average waiting time, average hop count). It rather degrades the performance as seen by occasional spikes in the average waiting time (Fig. 11) and in the average hop count (Fig. 12). PD evolves to improve the energy gain (Fig. 10), indicating that it successfully inhibits excessive reproduction through evolution. This results in the decreased population size (Fig. 13). ES evolves to improve the average hop count (Fig. 12), indicating that agents of this type spreads over platforms as a result of the evolution in their migration behavior. WA evolves to significantly improve the average energy gain (Fig. 10), indicating that it successfully reduces the number of random migration through evolution.

Figs. 10–12 also show that, independent of the type of agents that the simulation started with, agents in the simple network environment evolve to achieve the same level of performance as time progresses.

C. Evolution in a Dynamic Network

The second set of simulations (Figs. 14–21) evaluates the ability of the agents to adapt to dynamic network environments. Three types of dynamic network environments (i.e., a network with varying platform cost, a network with varying workload, and a network with platform failure) are considered in this set of simulations to evaluate how agents evolve and adapt to each environment. The following simulations start with the agent type PR (described in the previous subsection) and compare four performance measures (average energy gain, average waiting time, average hop count, and the agent population size) with and without the evolutionary adaptation mechanism.

1) A Network With Varying Platform Cost: In the simulations for a network with varying platform cost, each platform varies the resource cost depending on the number of agents that reside on the platform. Namely, the resource cost of a platform is determined by $N^2/3$, where N is the number of agents on the platform. Four platforms are randomly selected at the beginning of the simulation, and these platforms generate service requests at a constant rate of 25, 50, 75, and 100 service requests per second, respectively, throughout the simulation. The service time at an agent (i.e., time that an agent spends providing a service for a service request) is a constant 0.2 s, the same amount of time as in the simulations for a simple network. Other parameter values are also the same as those assumed in the previous subsection (i.e., those in Table IV).

Figs. 14–17 show the simulation results without the evolutionary adaptation mechanism, and Figs. 18–21 show the

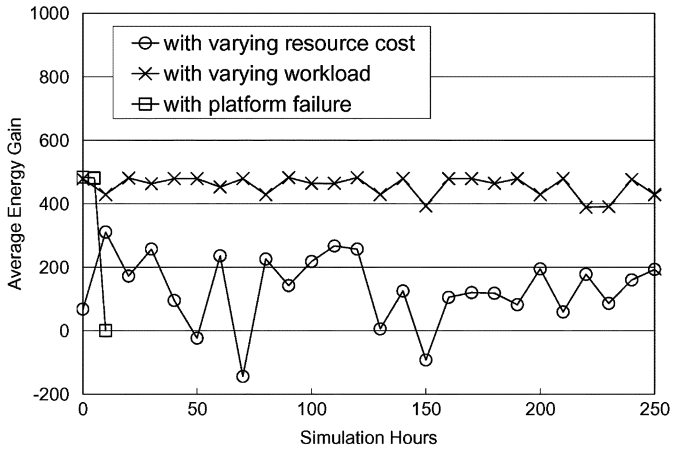


Fig. 14. Average energy gain without evolutionary adaptation.

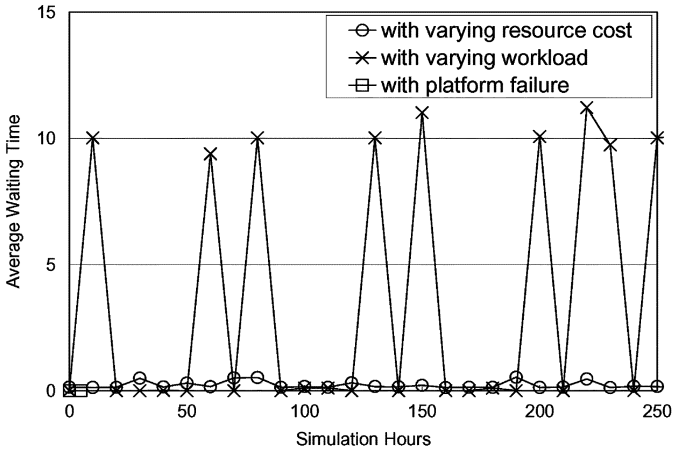


Fig. 15. Average waiting time without evolutionary adaptation.

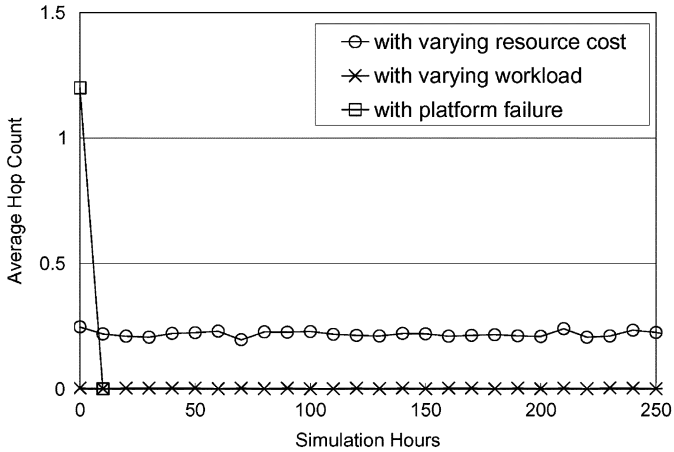


Fig. 16. Average hop count without evolutionary adaptation.

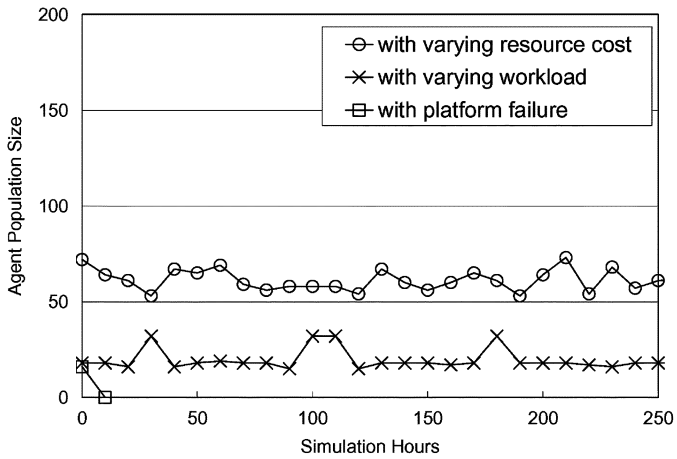


Fig. 17. Agent population size without evolutionary adaptation.

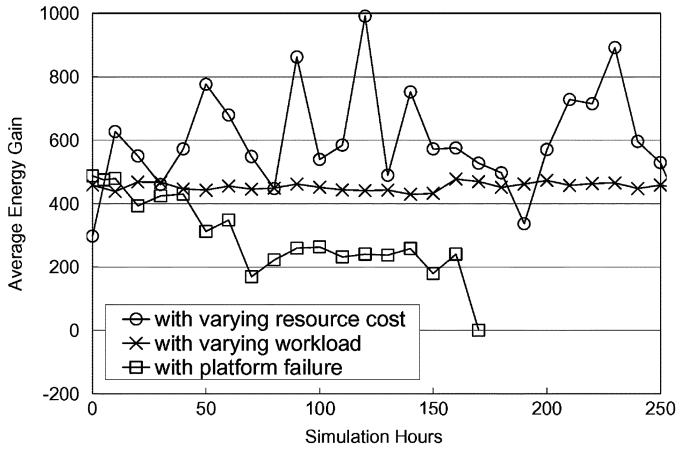


Fig. 18. Average energy gain with evolutionary adaptation.

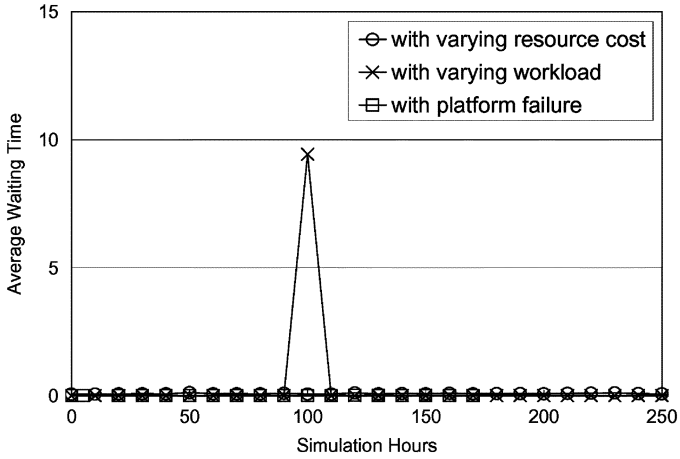


Fig. 19. Average waiting time with evolutionary adaptation.

simulation results with the evolutionary adaptation mechanism. Fig. 14 shows that, without the evolutionary adaptation mechanism, agents suffer from a significant loss in the energy gain, indicating that agents expend energy by staying at platforms whose resource cost is expensive. Fig. 18 shows that with the evolutionary adaptation mechanism, agents improve in the energy gain through evolution in their migration behavior. Figs. 15 and 19 show that the average waiting time improves

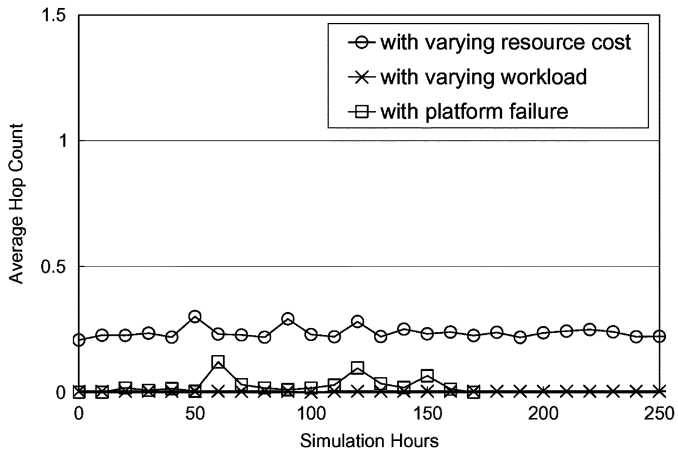


Fig. 20. Average hop count with evolutionary adaptation.

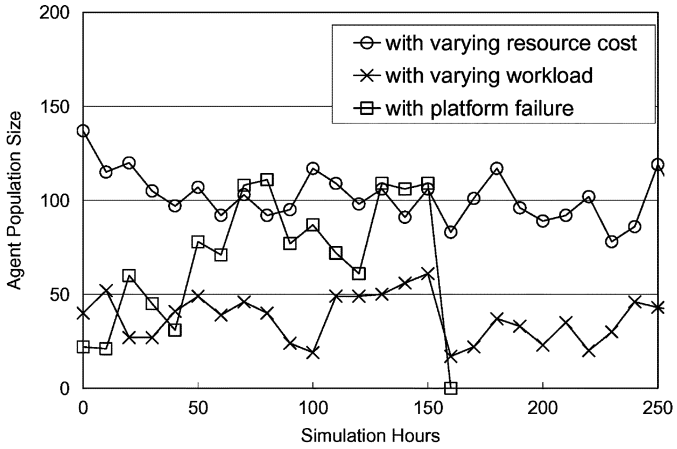


Fig. 21. Agent population size with evolutionary adaptation.

with the evolutionary adaptation mechanism.[1] This is because of the increased agent population size that results from evolution in their replication and reproduction behaviors, as shown in Figs. 17 and 21. The average hop count remains virtually the same with and without evolution (Figs. 16 and 20). As shown in the previous simulation results, the evolutionary adaptation mechanism allows agents to evolve to adapt to platform costs.

2) Network With Varying Workload: In the simulations presented so far, both the generation rate of service requests and service time of a request (i.e., time that an agent spends servicing a request) are constant. The following simulations consider a network with varying workload such that a request's service time varies. Ten platforms are randomly selected at the beginning of simulations, and each of these 10 platforms generates service requests at a constant rate of five service requests per second throughout the simulation. The service time of a request is randomly decided between 0.2 s and 0.4 s.

Fig. 15 shows that agents without the evolutionary adaptation mechanism suffer from large waiting times. On the other hand, agents with the evolutionary adaptation mechanism adapt

[1]Although it is difficult to see on Fig. 15 because of the scale of the vertical axis, for the agents without the evolutionary adaptation mechanism, the average waiting time over the simulation duration is 0.27 s, whereas, for the agents with the evolutionary adaptation mechanism, it is 0.08 s, approximately a 70% reduction. This clearly shows that the average waiting time improves with the evolutionary adaptation mechanism.

to the change in the workload to achieve small waiting times (Fig. 19). This is because the agents evolve to replicate and reproduce more (as seen in the increased agent population size in Figs. 17 and 21). The average hop count remains virtually the same with and without evolution (Figs. 17 and 21). As shown in the previous simulation results, the evolutionary adaptation mechanism allows agents to evolve to adapt to the dynamic changes in the workload.

3) A Network With Platform Failure: In the simulations for a network with platform failure, platforms may fail, and as a result, availability of platforms dynamically changes according to time. When a platform fails, it becomes unavailable for a certain time period (set to be 15 s in the simulations) before it becomes available again. When a platform fails, all agents that reside on the failed platform are removed from the system, as well as the service requests that the failed platform stored in its queue. When the simulation starts, availability of a platform (i.e., the probability that a platform is available) is 1.0. As time progresses, availability of a platform progressively decreases according to the following equation: $1.0-0.0025\times H$, where H is simulation hours. Five platforms are randomly selected at the beginning of simulations, and each of these five platforms generates service requests at a constant rate of 10 service requests per second throughout the simulation. The time to service a request is constant and is 0.2 s.

Figs. 14–21 demonstrate improved survivability as a result of the evolutionary adaptation mechanism. Without the evolutionary adaptation mechanism, agents become extinct in 10 simulation hours and become unable to continue providing a service (Figs. 14–17). On the other hand, with the evolutionary adaptation mechanism, agents evolve to survive significantly longer to maintain a certain level of the energy gain (Fig. 18) and the population size (Fig. 21), and to achieve small average waiting times (Fig. 19) and average hop counts (Fig. 20). This evolutionary adaptation occurs as a result of agents evolving toward dispersing among platforms and replicating and reproducing more often to reduce the risk of extinction in the presence of platform failure.

IV. Conclusion

This paper presents a framework to develop adaptive and scalable network services. Agents in the framework proposed in this paper are self-organizing and autonomous, and evolve. The attempt by the authors of this paper at applying the principles of self-organization and evolutionary adaptation to the design of the network services has produced promising results. Simulation results presented in this paper have shown that network services created by the framework autonomously adapt to network environments. To further address the evolutionary adaptation aspect of the framework, the authors of this paper are currently extending the simulation model to incorporate various methods proposed in the existing evolutionary computation literature [20].

In addition to the evolutionary adaptation aspect described in this paper, some key aspects of the framework proposed in this paper have been investigated. For instance, the proposed framework has been implemented and empirically evaluated in [26]–[28] to show that the overhead of the framework is comparable to that of existing mobile agent platforms. Some network services have also been implemented using the framework in [9]–[11].

The research in this paper, as well as the existing research on the framework proposed in this paper, presents a first step toward achieving the ultimate goal of removing human intervention from managing and tuning network services.

References

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford, U.K.: Oxford Univ. Press, 1999.

[2] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *J. Artif. Intell. Res.*, vol. 9, pp. 317–365, 1998.

[3] M. Dorigo and G. D. Caro, "Ant colony optimization: A new meta-heuristics," in *Proc. Congr. Evolutionary Computation*, 1999, pp. 1470–1477.

[4] J. L. Denebourg, S. Goss, N. Franks, A. SendovaFranks, C. Detrain, and L. Chretien, "The dynamics of collective sorting robot-like ants and ant-like robots," in *Proc. 1st Conf. Simulation of Adaptive Behavior: From Animal to Animats*, 1991, pp. 356–365.

[5] Y. Chen, R. H. Katz, and J. D. Kubiatowicz, "Dynamic replica placement for scalable content delivery," in *Proc. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2002, pp. 306–318.

[6] C. A. Coello, "A short tutorial on evolutionary multiobjective optimization," in *Proc. 1st Int. Conf. Evolutionary Multi-Criterion Optimization*, 2001, pp. 21–40.

[7] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proc. ACM SIGCOM*, 2002, pp. 177–190.

[8] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[9] T. Itao, T. Nakamura, M. Matsuo, T. Suda, and T. Aoyama, "Service emergence based on relationship among self-organizing entities," in *Proc. IEEE/IPSJ Symp. Applications and the Internet (SAINT)* , 2002, pp. 194–203.

[10] T. Itao, S. Tanaka, T. Suda, and T. Aoyama, "A framework for adaptive UbiComp applications based on the Jack-in-the-Net architecture," in *Kluwer/ACM Wireless Netw. J.*, vol. 10, 2004, pp. 287–299.

[11] T. Itao, S. Tanaka, and T. Suda, "SpaceGlue: Linking spaces for adaptive and situational service location," in *Proc. Int. J. Digital Libraries (JDL) Special Issue for Int. Conf. Service-Oriented Computing 2003*.

[12] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[13] J. R. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[14] R. C. Kube and H. Zhang, "Collective robotics: From social insects to robots," *Adapt. Behav.*, vol. 2, no. 2, pp. 189–218, 1994.

[15] H. T. Kung and C. H. Wu, *Content Networks: Taxonomy and New Approaches*, K. Park and W. Willinger, Eds. Oxford, U.K.: Oxford Univ. Press, 2003. The Internet as a Large-Scale Complex System.

[16] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," in *Proc. Web Caching and Content Distribution Workshop (WCW'01)*, 2001.

[17] W. Y. Ma, B. Shen, and J. T. Brassil, "Content services networks: The architecture and protocol," in *Proc. 6th Int. Workshop on Web Caching and Content Distribution*, 2001.

[18] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[19] T. Nakano and T. Suda, "Adaptive and evolvable network services," in *Proc. Genetic and Evolutionary Computation Conf.*, vol. I, 2004, pp. 151–162.

[20] ——, "Applying biological principles to the design of network services," Appl. Soft Comput. J., to be published.

[21] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*: O'Reilly, 2001.

[22] G. Pierre, M. van Steen, and A. Tanenbaum, "Dynamically selecting optimal distribution strategies for web documents," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 637–651, Jun. 2002.

[23] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *Proc. IEEE INFOCOM*, 2001, pp. 1587–1596.

[24] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A dynamic object replication and migration protocol for an Internet hosting service," in *Proc. Int. Conf. Distributed Computing Systems*, 1999, pp. 101–113.

[25] R. Schoonderwoerd, O. E. Holland, J. L. Bruten, and L. J. M. Rothkrantz, "Ant-based load balancing in telecommunications networks," *Adapt. Behav.*, no. 2, pp. 169–207, 1996.

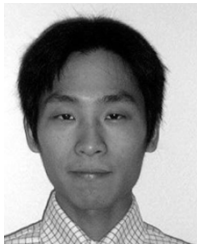
[26] S. Sameshima, J. Suzuki, S. Steglich, and T. Suda, Platform independent model (PIM) and platform specific model (PSM) for super distributed objects, in Object Management Group, Final Adopted Specification, OMG Document Number: dtc/03-09-01, Sep. 2003.

[27] J. Suzuki and T. Suda, "Design and implementation of an scalable infrastructure for autonomous adaptive agents," in *Proc. 15th IASTED Int. Conf. Parallel and Distributed Computing and Systems*, Marina Del Rey, CA, Nov. 2003.

[28] ——, "A middleware platform for a biologically-inspired architecture supporting autonomous and adaptive network applications," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 127–146, Feb. 2005.

[29] T. Suda, T. Itao, and M. Matsuo, "The bio-networking architecture: The biologically inspired approach to the design of scalable, adaptive, and survivable/available network applications," in *The Internet as a Large-Scale Complex System*, K. Park, Ed. Oxford, U.K.: Oxford Univ. Press, 2005. to appear.

[30] X. Yao, "Evolving artificial neural networks," in *Proc. IEEE*, vol. 87, 1999.



Tadashi Nakano (M'05) received the B.E., M.E., and Dr.E. degrees in information systems engineering from Osaka University, Osaka, Japan, in 1999, 2000, and 2002, respectively.

Since 2002, he has been with the School of Information and Computer Science, University of California, Irvine, where he is researching a biologically inspired networking architecture as a postdoctoral research scholar. His current research interests include biologically inspired networking and molecular communication for nanomachines.

Dr. Nakano is a Member of the Association for Computing Machinery (ACM).

Tatsuya Suda (S'80–M'82–SM'97–F'01) received the B.E., M.E., and Dr.E. degrees in applied mathematics and physics from Kyoto University, Kyoto, Japan, in 1977, 1979, and 1982, respectively.

From 1982 to 1984, he was with the Department of Computer Science, Columbia University, New York, as a Postdoctoral Research Associate. Since 1984, he has been with the Department of Information and Computer Science, University of California, Irvine, where he is currently a Professor. He has also served as a program director of the Networking Research Program at the National Science Foundation from 1996 to 1999. He received an IBM postdoctoral fellowship in 1983. He was a visiting associate professor at the University of California, San Diego, a Hitachi Professor at the Osaka University and currently is a NTT Research Professor. He is an Area Editor of the *International Journal of Computer and Software Engineering*. He is a member of the Editorial Board of the *Encyclopedia of Electrical and Electronics Engineering*.

Dr. Suda was the Conference Coordinator from 1989 to 1991, the Secretary and Treasurer from 1991 to 1993, the Vice Chairman from 1993 to 1995, and the Chairman from 1995 to 1997 of the IEEE Technical Committee on Computer Communications. He was also the Director of the U.S. Society Relations of the IEEE Communications Society from 1997 to 1999. He is an Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING, a Senior Technical Consultant to the IEEE TRANSACTIONS ON COMMUNICATIONS, a former Editor of the IEEE TRANSACTION ON COMMUNICATIONS. He was the Chair of the 8th IEEE Workshop on Computer Communications and the TPC co-chair of the IEEE Infocom 97. He is a Member of the Association for Computing Machinery (ACM).