Adaptive and Evolvable Network Services

Tadashi Nakano and Tatsuya Suda

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{tnakano, suda}@ics.uci.edu

Abstract. This paper proposes an evolutionary framework where a network service is created from a group of autonomous agents that interact and evolve. Agents in our framework are capable of autonomous actions such as replication, migration, and death. An evolutionary mechanism is designed using genetic algorithms in order to evolve the agent's behavior over generations. A simulation study is carried out to demonstrate the ability of the evolutionary mechanism to improve the network service performance (e.g., response time) in a decentralized and self-organized manner. This paper describes the evolutionary mechanism, its design and implementation, and evaluates it through simulations.

1 Introduction

Swarm intelligence – the collective intelligence of groups of simple individuals often observed in social insects [1,8] has inspired many applications in a variety of fields such as optimization [5,9], clustering [4], communication networks [17] and robotics [10]. In these applications, individuals are capable of sensing the environment in which they operate, and act based only on partial information about the entire environment. Each individual is designed simply and does not have the ability to accomplish a goal. However, a collection of individuals exhibits intelligent behavior toward achieving a goal along with useful properties such as adaptability and scalability.

This paper proposes an evolutionary framework for developing distributed network services that require a large number of network components (e.g., data and software) to be replicated, moved and deleted in a decentralized manner. Such network services may include content distribution networks [2,7,14,15,16], content services networks [11] and peer-to-peer file sharing networks [13]. Analogous to those applications inspired by swarm intelligence, agents are simply designed using only local information without relying on global knowledge, and collectively provide adaptive and scalable network services.

In the proposed framework, a single network service is provided by a group of autonomous agents. Each autonomous agent implements an identical network service (e.g., a content hosting service or a web document), but can have different behavior in replication, migration and death (deletion of itself). An agent's behavior is governed by a set of genes embedded into each agent, and designed to evolve through a repro-

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 151–162, 2004.

ductive process with genetic algorithms [12]. As opposed to generational genetic algorithms that rely on god-like central selection, agents in our framework are evaluated and selected by nearby agents in a decentralized and self-organized way.

A simulation study is carried out to present the ability of evolutionary adaptation to improve the service performance or fitness values (e.g., response time, bandwidth consumption, resource usage, etc.) It is shown through evolutionary adaptation processes that agents evolve their behavior over generations by which a network service becomes adapted to a variety of network environments.

The rest of the paper is organized as follows. Section 2 gives a broad overview of the proposed evolutionary framework, and then presents the design of agents and their evolutionary mechanisms. To evaluate the evolutionary mechanisms, preliminary simulations are run in Section 3 and an extensive set of simulation studies is carried out in Section 4. Section 5 concludes the paper with a brief summary.

2 Evolutionary Framework for Developing Network Applications

This section first provides an overview of the proposed framework for developing distributed network applications and then describes the evolutionary mechanisms designed for network applications to adapt to network environments.

2.1 Overview

Our framework assumes a fully distributed network environment in which a group of autonomous agents self-organizes a network service without centralized control. It is also assumed that communications and information are restricted to those locally available to agents (e.g., agents can communicate with each other only when residing on the same network platform or on adjacent ones.)

As illustrated in Figure 1, our network is modeled using three network components: agents, users and platforms, which exchange a common resource called *energy*.

Agents provide a service to users (i.e. end service consumers or an agent of an application) in exchange for energy. Agents use computing resources (e.g., CPU power, memory, and network bandwidth) provided by a hosting platform in exchange for energy. Agents also use energy to invoke behaviors such as replication, migration and death. Thus, the energy level of an agent is a measure of how efficiently the agent provides a service, uses computing resources and performs behavior.

The platforms are autonomous systems connected to each other. Platforms host agents and provide computing resources such as CPU, memory and bandwidth. Platforms provide an environment for agents where agents can migrate and replicate. Platforms periodically charge agents for energy, and expel agents who run out of energy. Thus, platforms perform natural selection and favor energy efficient agents.

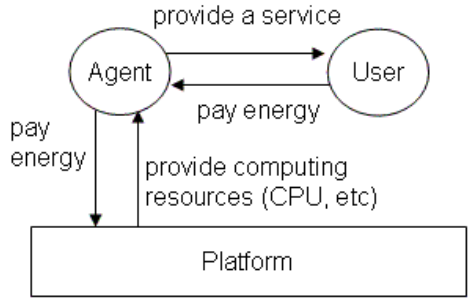

Fig. 1. Energy exchange

2.2 Agents and Their Behavior

Agents have an internal state that affects behaviors such as replication, migration and death. Example internal states include *energy intake* (the difference between acquired energy units and consumed energy units), *age* (time elapsed since birth) and *activeness* (the degree of willingness to invoke behavior). The internal state also includes an agent's performance directly related to a network service, such as *response time* (time taken for a user to receive a service from the agent).

Agents are capable of sensing the local environmental conditions of the platform that they reside on and also the environmental conditions of adjacent platforms. Environmental conditions include *request rate* (how often the platform that hosts an agent receives requests from a user), *request rate change* (how much the request rate increases or decreases), *population* (the number of agents on the platform), *resource cost* (the energy cost of resource at the platform), *behavior cost* (the energy cost of behavior at the platform).

According to an environmental condition or their internal state described above, agents autonomously replicate, reproduce, migrate or die. For instance, agents may replicate (make a copy of themselves) or reproduce (produce an agent with another agent) in response to an increased demand for the service; agents may migrate from one platform to another to perform a service in the vicinity of users; agents may die when the service they provide becomes outdated.

Figure 2 illustrates the behavior invocation mechanism embodied in each agent. In replication, reproduction and death, a set of input values (V_i) that includes internal states and environmental conditions described above is multiplied with a set of associated weights (W_i). If the weighted sum $(\Sigma V_i \times W_i)$ exceeds the threshold (θ), then a corresponding behavior is invoked. In migration, agents need to choose which platform to migrate to from the current platform. In this case, the equation shown in Figure 2 is examined for all the possible choices of platforms including the current platform. If multiple choices satisfy the equation, the behavior most likely triggered is the choice that produces the highest sum. If the current platform produces the highest sum, the agent remains on the current platform.

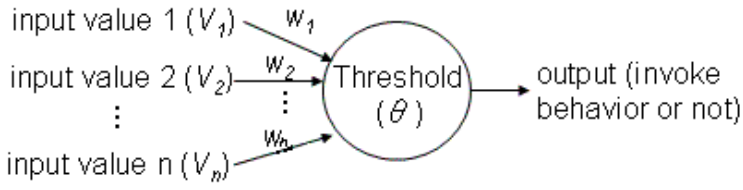

Fig. 2. Behavior invocation mechanism

2.3 Evolutionary Adaptation Mechanisms

In our framework, evolution occurs as a result of selection from diverse behavioral characteristics of individual agents. Agents have a set of genes (weights) that governs their behavior, and diverse behavioral agents arise from the genetic variation. When an agent reproduces, the agent selects a reproduction partner so as to improve one of the service performances (e.g., response time, bandwidth consumption, resource usage). Meanwhile, the diverse behavioral characteristics among agents are generated by genetic operators (mutation and crossover). Through successive generations, beneficial features are retained while detrimental behaviors become dormant, enabling the network service to adapt to the network environment. The following explains the evolutionary mechanism designed in this paper.

Representation. Each agent is represented as a vector of real values (a set of genes) with each real value corresponding to a weight shown in Figure 2.

Natural selection. Agents who run out of energy are eliminated by a platform. This mechanism is referred to as natural selection. Natural selection guarantees that inefficient behavioral agents, (which may be produced through the genetic operators such as crossover and mutation,) eventually become extinct.

Partner selection. In reproduction, an agent selects a partner agent on the same platform or adjacent platforms according to the following fitness assignment strategy that involves multiple fitness values including (1) *waiting time* (the average interval between the time that a service request arrives at a platform and the time that an agent provides the requested service), (2) *hop count* (the average number of platform hops that a service request travels from a user to the agent), and (3) *energy efficiency* (the fraction of the consumed energy and acquired energy). In the following, each of these fitness values is explained with more detail.

Waiting time is defined as the average interval between the time that a service request arrives at a platform and the time that an agent provides the requested service. Upon arrival at a platform, a service request is placed on a platform queue if all agents on the platform are busy processing other service requests. In order to improve this fitness value, agents are required to reproduce when there is insufficient number of agents to handle all service requests from users. Agents may further improve the waiting time by reproducing in advance when the number of service requests starts increasing.

Hop count is measured as the average number of platform hops that a service request travels from a user to the agent. It is assumed that a service request issued by a user is always directed to one of the agents nearest to the user. Agents can improve the hop count by staying around users or following users if they are moving around.

Energy efficiency is measured as the fraction of the consumed energy and acquired energy. The acquired energy is the total amount of energy that the agent obtains from its birth, and consumed energy is the total amount of energy that the agent consumes from its birth. The amount of acquired energy increases in proportion to the number of service requests that the agent processes. Consumed energy represents how efficiently the agent performs behavior and uses computing resources. For instance, inefficient behavior invocation (e.g., migrating too often) may incur a great energy loss. In order to improve energy efficiency, agents need to balance or even optimize its energy income relative to energy expenditure.

To select a reproduction partner, an agent probabilistically determines which fitness value needs improvement: waiting time, hop count or energy efficiency. The fitness value chosen for possible improvement is based on the agent's own fitness values and pre-defined desired fitness values (initially given to all agents) with respect to these three criteria.

Specifically, given the set of its own fitness values (F_i), the set of predefined required fitness values (R_i), the following equation defines the probability of the fitness value j being chosen: $(R_j/F_j)/\Sigma(R_i/F_i)$. Suppose that an agent is inefficient in satisfying a pre-defined level of energy efficiency, and then the agent is likely to choose the energy efficiency as the fitness value to improve in its reproduction. Similarly, the agent may choose the waiting time or hop count as the fitness value when it is inefficient in satisfying either of the pre-defined desired fitness values.

After selecting the fitness, candidate agents are ranked according to the selected fitness value through linear rank selection typically used in evolutionary computation [12].

Crossover. After an agent selects a reproduction partner, the two sets of weights from the two parent agents are crossed over to produce a new set of weights for a child agent. In crossover, a set of weights for a child agent is determined in such a way that more weights are inherited from the parent with a greater fitness value. The probability of a weight being chosen from a parent linearly increases in proportion to the number of its fitness values that are greater than those of the other parent.

Mutation. After two parental weights are crossed over, mutation may occur at each weight of a child agent with a probability called mutation rate. In mutation, each weight value is subject to random change within a certain range called mutation range.

The design of evolutionary mechanisms that were described in this section is an example, and alternative approaches developed in the evolutionary computation literature such as multiobjective optimization [3] and adaptive parameter control [6] are also applicable to implementing our agent-based evolutionary framework.

3 Simulations in Static Network Environments

In this section, the evolutionary mechanism designed in Section 2 is evaluated in relatively simple and homogenous network environments through a simulation study. Pseudo code of the simulator algorithm is shown in Figure 3. The various parameters used and simulation results are explained in the following.

3.1 Configurations

A simulated network is configured as an 8×8 mesh topology network with 64 nodes. Each node on the network hosts a single platform that charges each agent 1 energy unit per second for computing resources (one simulation cycle corresponds to 1 second). There are seven users in the network. Each user generates service requests at different rates ranging from 10 to 30 requests per second, totaling 150 requests per second on the entire network. A service request issued by a user is forwarded to one of the nearest platforms where agents exist. After forwarding, if no agents are available for processing the service request, the service request is placed on the platform queue and experiences a delay until an agent becomes available and processes it.

The simulations assume agents that are capable of reproduction, migration and death. In addition, each agent can process a maximum of 5 service requests per second. An agent receives 10 energy units from a user in exchange for each request processed. Every 15 seconds an agent makes a decision on whether to invoke death, reproduction, and migration. Agents consume 500 energy units for performing behavior such as reproduction and migration. Agents may invoke a single behavior (one out of

```
Initialize platforms, agents and users
While (not simulation last cycle)
        For each user do
                send service requests to one of the nearest agents according to
                configured service request rates.
        End For
        For each platform do
                charge agents platform cost
        End For
        For each agent do
                If received service requests do
                        Process the requests and receive energy
                End If
                make decision on reproduction, migration and death
                update average waiting time, hop count, energy efficiency
        End For
End While
```

Fig. 3. Pseudo code of simulation algorithm

Table 1. Three types of agents used in simulations: PR, PD and ES

Weight	PR (primary)	PD (productive)	ES (energy seeker)
R. Request Rate	0.1	0.3	0.1
R. Threshold	0.5	0.5	0.5
M. Request Rate	1.0	1.0	2.0
M. Resource Cost	0.5	0.5	0.0
M. Population	1.0	1.0	0.5
M. Activeness	0.5	0.5	0.5
M. Threshold	2.0	2.0	2.0

three) or multiple behaviors (reproduction and migration), or agents may decide not to invoke any behaviors according to a behavior invocation mechanism shown in Figure 2.

In reproduction, agents select a partner agent that is older than 30 simulated minutes. Agents replicate when there are no partner agents. In both reproduction and replication, a parent agent (the parent who decides to reproduce, not the parent selected as a reproduction partner) provides 3000 energy units to its child agent. Mutation occurs for each weight probabilistically according to the mutation rate of 0.2. In mutation, each weight is subject to random change within a range of mutation range of 0.1. Crossover and mutation are applied only to weights and not applied to thresholds. The predefined performance requirement used in partner selection is 1.0 seconds for the average waiting time, 0.5 hops for the average hop count, and 0.3 for the average energy efficiency.

In migration, agents are allowed to migrate only to adjacent nodes linked with the current node. Reproduction is allowed between two agents on the same platform or on adjacent platforms. Agents die when they exhaust their energy or reach a maximum age of 4 simulated hours.

3.2 Simulation Results

One of the three types of agents listed in Table 1 (See Section 2.2 for the detail of weight) is placed on the top left corner of the network, given 10000 energy units, and simulations are run for 10 simulated days.

Figures 4 through 6 depict simulation results for each of the four types of agents, comparing the performance of agents with the evolutionary mechanism against that of agents without the evolutionary mechanism. Figure 7 shows the dynamics of an agent population. In these figures, the horizontal axis indicates simulation time (in 1 hour increments), while the vertical axis indicates the total energy gain of all agents, average waiting time of all service requests, average hop count of all service requests, the number of agents.

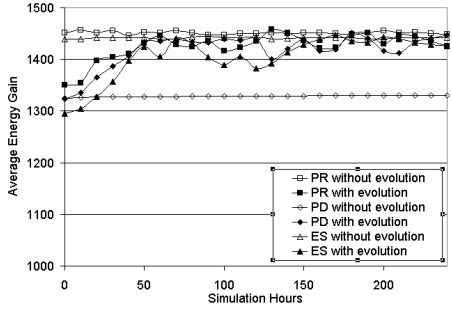

Fig. 4. Energy gain

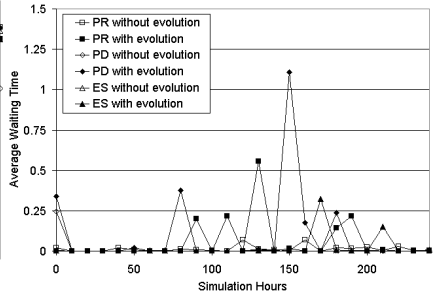

Fig. 5. Average waiting time

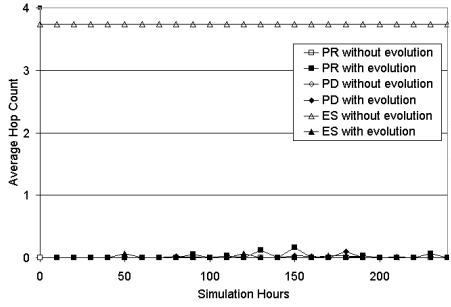

Fig. 6. Average hop count

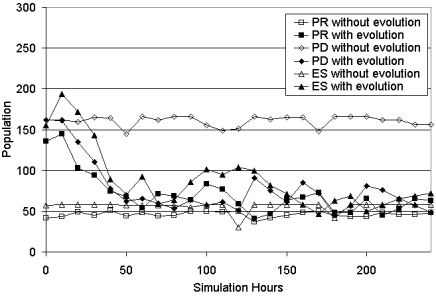

Fig. 7. Population dynamics

These simulation results are summarized as follows; PR doesn't evolve to improve any of the performance criteria considered, but rather degrades the performance; PD evolves toward invoking replication and reproduction less often, and improves the energy efficiency; ES evolves toward migrating to a user requesting a service and significantly improves the average hop count.

The simulation results explained above show that the evolutionary mechanism designed in this paper successfully improves performance, e.g., energy gain of PD, hop count of ES. However, the performance improvement comes with an overhead associated with the constantly repeating evolutionary process. In other words, when agents achieve a sufficiently low waiting time and hop count (e.g., the both averages reach nearly 0), they then try to minimize energy consumption. This means that in reproduction they select a partner agent based on energy efficiency (e.g., an agent who invokes reproduction or migration less often). This energy efficiency based selection is constantly performed, which creates a situation in which there are not enough agents to process all service requests on the network, resulting in occasional spikes in waiting time and hop count. A possible improvement would be brought about by applying an adaptive evolution technique traditionally suggested in the evolutionary computation literature [6], e.g., by reducing the mutation rate as solutions become closer to optima.

4 Evolution in a Variety of Network Environments

This section describes the extensive set of simulations performed in a variety of network environments and demonstrates the adaptability of the network application with evolutionary mechanisms.

4.1 Simulation Configurations

Evolutionary mechanisms are evaluated in the following three kinds of network environments. Note that each of these networks is a modified version of the static network used in the previous simulation and, unless otherwise stated, simulation configurations (e.g., the network topology, size, platform cost, etc) are the same as the previous ones. Also, only the PR agent (shown in Table 1) is used in the following simulations.

Network with varying resource cost: All platforms vary the resource cost depending on how many agents they are hosting. The following formula is experimentally used to determine the resource cost: *resource_cost = (the number of agents)×2/3*.Four users are placed on four different platforms. Each user has a different service request generation rate, 25, 50, 75 and 100 service requests per second.

Network with varying workload: A user requests either of two types of services: one service that takes agents 0.2 seconds to process, or another that takes 0.4 seconds. There are 10 users, who issue 5 service requests per second, randomly switching the types of services to request.

Network with platform failures: All platforms have a possibility of failure. Platform failures destroy all agents residing on the failed platform, and any service request in the platform queue is also discarded. The availability of all platforms starts with 1.0 and progressively decreases based on the following equation: *platform_availability = 1.0 – 0.025×simulation_hours*. Platforms probabilistically fail every single minute based on platform availability, and the failed platforms become available one second later. Five users are placed on five randomly selected platforms. The users stay on the selected platforms and generate 10 service requests per second throughout the simulations.

4.2 Simulation Results

Simulations are run with and without evolutionary mechanisms in each of the three kinds of networks. Results are shown in figures 8 through 11 and explained in the following.

In the network with varying resource cost, agents without the evolutionary mechanism suffer from a great energy loss leading to lower populations on a platform, and the inadequate number of agents leads to higher waiting time. Evolutionary adaptation allows agents to adapt to the locality of the network environment: those agents on expensive platforms evolve toward migrating away from the platforms, and those who

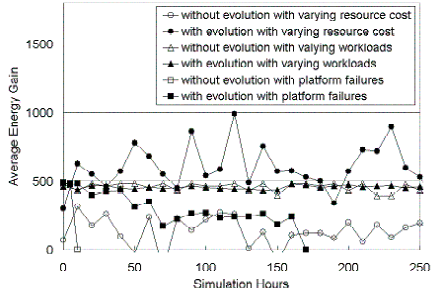

Fig. 8. Energy gain

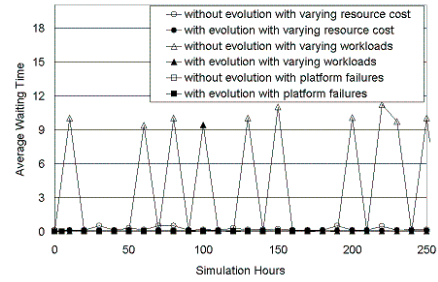

Fig. 9. Average waiting time

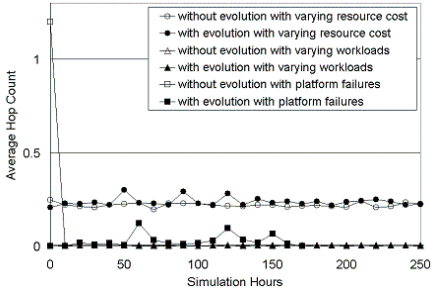

Fig. 10. Average hop count

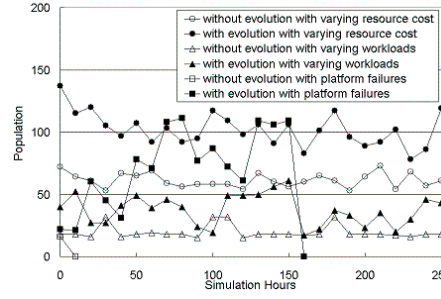

Fig. 11. Population dynamics

are on the affordable platform remain there. Consequently, the average hop count slightly increases, but the average energy gain significantly improves.

In the network with varying workload, agents without evolutionary adaptation severely increase the waiting time when the type of a service requested is changed. On the other hand, evolving agents appear to be responsive to the change: when experiencing a high waiting time, responsive agents that exhibit higher reproduction rate are likely to be selected as reproduction partners, which accelerates reproduction, resulting in a reasonable average waiting time.

In the network with platform failures, without the evolutionary mechanism, agents become extinct after consecutive platform failures and are unable to continue providing the network application. On the other hand, the evolutionary mechanism leads to population increase and dispersal, resulting in more available and robust network applications. This evolutionary adaptation occurs because agents that can distribute among platforms and that can produce more agents are more likely to survive in the face of platform failure.

5 Conclusions

This paper proposes an evolutionary framework where a network service is created from a group of simple agents that interact and evolve. This work has been done in

order to improve the emergent properties such as scalability and adaptability of the Bio-Networking Architecture [18], which is a generic framework for building large-scale network services based on biological principles.

A design concept similar to the one applied to our framework is found in swarm intelligence [1,8], where simple individual agents collectively solve a problem. Swarm intelligence has been applied to some networking issues such as network routing and load-balancing [17], and we further apply the swarm intelligence to build adaptive and scalable network applications.

Although specific applications of our framework are not discussed in this paper, autonomous agents in our framework can represent any network objects (data and a service), and thus our framework can be applied, for instance, to replica placement of network objects for CDNs (Content Distribution Networks) and P2P networks. As opposed to replica placement algorithms proposed for CDNs and P2P networks [2,7,14,15,16], which are often complex, the behavior algorithm of agents in our framework is simple and easy to design, yet demonstrates through simulations that the service performance improves with evolutionary adaptation. An additional simulation study will be done focusing on specific network services or applications. In addition, the presented simulation study assumes a network that contains only one network service. Ongoing work is extending this model to include multiple network services that interact and coevolve.

Acknowledgements. This work was supported by the National Science Foundation through grants ANI-0083074 and ANI-9903427, by DARPA through Grant MDA972-99-1-0007, by Air Force Office of Scientific Research through Grant MURI F49620-00-1-0330, and by grants from the University of California MICRO Program, Hitachi, Hitachi America, Novell, Nippon Telegraph and Telephone Corporation (NTT), NTT Docomo, Fujitsu, and NS Solutions Cooperation.

References

1. E. Bonabeau, M. Dorigo and G. Theraulaz, “Swarm intelligence: from natural to artificial systems,” Oxford University Press, 1999.
2. Y. Chen, R. H. Katz and J. D. Kubiatowicz, “Dynamic replica placement for scalable content delivery,” in Proceedings of the First International Workshop on Peer-to-Peer Systems, pages 306-318, 2002.
3. C. A. Coello Coello, “A short tutorial on evolutionary multiobjective optimization,” First International Conference on Evolutionary Multi-Criterion Optimization, Springer-Verlag, Lecture Notes in Computer Science, No. 1993, pages 21-40, 2001.
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 Proceedings of the 1st Conference on Simulation of Adaptive Behavior: From Animal to Animats, MIT Press, pp. 356-365.

5. M. Dorigo and G. D. Caro, “Ant algorithms for discrete optimization”, in Proceedings of the Congress on Evolutionary Computation, 1999.
6. A. E. Eiben, R. Hinterding and Z. Michalewicz, “Parameter control in evolutionary algorithms,” IEEE Transactions on Evolutionary Computation,” Vol. 3, No. 2, pages 124-141, 1999.
7. M. J. Kaiser, K. C. Tsui and J. Liu, “Adaptive distributed caching,” in Proceedings of the IEEE Congress on Evolutionary Computation, pages 1810-1815, IEEE, 2002.
8. J. Kennedy and R. C. Eberhart, “Swarm intelligence,” Morgan Kaufmann Publishers, 2001.
9. J. Kennedy and R.C. Eberhart, “Particle swarm optimization,” in Proceedings of the IEEE International Conference on Neural Networks, Vol. 4, 1942-1948, 1995.
10. R. C. Kube and H. Zhang, “Collective robotics: from social insects to robots,” Adaptive Behavior, Vol. 2, No. 2, pp.189-218, 1994.
11. W. Y. Ma, B. Shen and J. T. Brassil, “Content services networks: the architecture and protocol,” in Proceedings of the 6th International Workshop on Web Caching and Content Distribution, 2001.
12. M. Mitchell, “An introduction to genetic algorithms,” MIT Press, 1996.
13. A. Oram, “Peer-to-Peer: harnessing the power of disruptive technologies,” O'Reilly & Associates, 2001.
14. G. Pierre, M. van Steen and A. Tanenbaum, “Dynamically selecting optimal distribution strategies for web documents,” IEEE Transactions on Computers, 51(6), 2002.
15. L. Qiu, V. N. Padmanabhan and G. M. Voelker, “On the placement of web server replicas,” in Proceedings of the IEEE INFOCOM 2001, pages1587-1596, 2001.
16. M. Rabinovich, I. Rabinovich, R. Rajaraman and A. Aggarwal, “A dynamic object replication and migration protocol for an Internet hosting service,” in Proceedings of the International Conference on Distributed Computing Systems, pages 101-113, 1999.
17. R. Schoonderwoerd, O. E. Holland, J. L. Bruten and L. J. M. Rothkrantz, “Ant-based load balancing in telecommunications networks,” Adaptive Behavior, Vol. 5, No. 2, MIT Press, pp.169-207, 1996.
18. 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 K. Park (ed.), The Internet as a Large-Scale Complex System, Oxford University Press, 2003.