Bio-inspired Organization for Multi-Agents on Distributed Systems

Ichiro Satoh

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail: ichiro@nii.ac.jp

Abstract. This paper presents a middleware system for multi-agents on a distributed system as a general test-bed for bio-inspired approaches. The middleware is unique to other approaches, including distributed object systems, because it can maintain and migrate a dynamic federation of multiple agents on different computers. It enables each agent to explicitly define its own deployment policy as a relocation between the agent and another agent. This paper describes a prototype implementation of the middleware built on a Java-based mobile agent system and its practical applications that illustrates the utility and effectiveness of the approach in real distributed systems.

1 Introduction

Distributed computing systems are composed of a number of computers. The scale and complexity of distributed modern systems are beyond centralized and hierachical management techniques. Distributed systems are dynamic in the sense that, computers may be added to or removed from it and channels between computers may be disconnected or changed. Software components, which an application consists of, are required to be adapted and deployed at computers in a distributed system according to changes in the requirements of applications and the structure and computational resources of the system.

This paper addresses the deployment of partitioned applications over a distributed system, because it is one of the most important issues regarding where and what software will be deployed at computers. It presents a framework to adapt a federation of software components. The framework is based on two key ideas. The first is to enable each component to specify its own deployment policy instead of any global policies. The second is to facilitate the dynamic federation of multiple components as more than one virtual distributed system over a real distributed system, instead of any simulation-based environments. The framework enables such a federation to be transformed and made mobile through bio-inspired self-organization, such as that undertaken by cells in their transforming and crawling locomotion. Furthermore, the framework can be used as a general test-bed for providing various bio-inspired approaches in distributed systems, as well as a middleware system for adaptive distributed systems.

Several researchers have attempted to introduce biological metaphors into distributed systems. Most of this work has been based on simulation-based approaches. For example, Swarm [6] and MASS [3] are general simulators for multi-agent models. However, real systems are complex and varied. Unfortunately, most existing simulation-based results seem to have been based on arbitrary hypotheses in the sense that various parameters in their simulations have lacked any technical grounds. Unfortunately, such unrealistic simulations have often only provided non-sensical or impractical results. We still lack a great deal of data that are essential to simulating the approaches accurately. Therefore, real experiments in distributed systems must have priority over simulation-based experiments for us to accumulate actual experience.

2 Approach

A distributed application consists of partitioned applications that may run on different computers. This paper assumes that each partitioned application, called an agent, can be autonomous and mobile. To adapt an application to changes in a distributed system, partitioned applications, i.e., agents, partitioned applications must not be bound to particular computers. They should be dynamically deployed at appropriate computers without any centralized management system.

This can be supported by an metaphors drawn from biological process. When a computer is removed from the system or it shuts down, agents running on it should escape. Lamellipodia are flattened and protrusive projections that periodically expand from the surface of a cell. Effective movement requires a motile cell to be polarized, so that its protoplasm membrane is relatively quiescent everywhere else except its leading edge where lamellipodia periodically project outward in all directions. As they pull on one another they create intervening regions in which the cortex is stretched. This tug-of-war continues until one lamellipodium aligns in a dominant direction and becomes unipolar, then migrates in that direction. Lamellipodia can be viewed in terms of speculative migration or expansion.

Each agent should be able to explicitly specify its own constraints to migrate agents. For example, if an agent has a migration constraint dependent on another agent, when the otwher agent moves to another location, the former agent decides its destination according to its own migration constraints, i.e., the source or destination of the other agent. Such constraints are defined as policies within agents and allow us to specify physical structures and mechanisms in motile cells, such as membrane and cytoplasmic streaming. We provide several policies for agents to support bio-inspired deployments of agents.

3 Design and Implementation

The framework presented in this paper is a middleware for deploying and executing general-purpose software components. It can be used as a general test-bed for providing various bio-inspired approaches, in particular bio-inspired deployment of software, in distributed systems. It was implemented in Java (J2SE version 1.4 or later versions) and agents are implemented as a set of Java objects.

3.1 Runtime system

Figure 1 outlines the basic structure of a runtime system. Each establishes at most one TCP connection to each of its neighboring hosts and exchanges control messages, agents, and inter-agent communications with the other runtime systems through the connection. Since it is constructed on the Java virtual machine, it can conceal differences between the platform architecture of the source and destination computers. All runtime systems can exchange agents with others through the use of mobile agent technology.

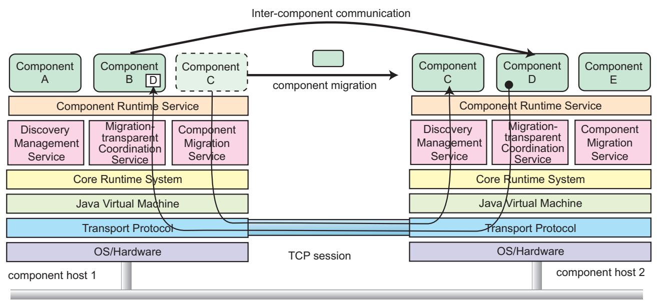

Fig. 1. Architecture of agent runtime system.

Each agent can itinerate between multiple computers under their own control [7–9]. After arriving at its destination or being duplicated, each agent can continue working without losing accumulated work, such as the content of instance variables in the agent's program, at the source computers. It is also equipped with its own identifier and that of the federation that it should belong to. It can explicitly specify the computational capability that its destination hosts must offer in CC/PP [12] form as we will discuss later. If an agent is on a computer that cannot satisfy its requirements, its intent is to leave computer. While each agent is running, it can declare at most one deployment policy and one or more message policies by invoking a built-in method of the class that every agent must inherit. Although policies are open for developers to define their own policies, the current implementation provides the following deployment policies.

- If an agent declares a *follow* policy for another agent, when the latter migrates to another computer, the former migrates to the latter's destination computer.
- If an agent declares a *dispatch* policy for another agent, when the latter migrates to another computer, a copy of the former is created and deployed at the latter's destination computer.
- If an agent declares a *shift* policy for another agent, when the latter migrates to another computer, the former migrates to the latter's source computer.
- If an agent declares a *fill* policy for another agent, when the latter migrates to another computer, a copy of the former is created and deployed at the latter's source computer.

Figure 2 outlines four deployment policies. These policies are related to phenomena in biological processes. For example, a `follow` policy enables an agent to come near another agent. For example, when multiple agents declare a policy for a leader agent, they can swarm around it. A *shift* policy enables an agent to follow the movement of another agent. The former agent can track the latter as it moves. The policy thus corresponds to the phenomenon of cytoplasmic streaming. A `dispatch)` policy enables an agent to stay in the current location and then deploy its clone at the destination of another moving agent. It can model the footprint of a motile cell. We have assumed that an agent can declare the policy for another agent and specify the TTLs of its clones as their life-spans. As the latter agent moves, cloned former agents are deployed at its footmark and these clones are automatically volatilized after their life-spans are over. Therefore, the clone agents can be viewed as a pheromone that is left behind after the latter agent has moved on. A *fill* policy corresponds to the phenomenon of cell division.

When an agent is created, the dispatch and fill policies can explicitly control whether the newly created agent can inherit the state of its original agent. The following message policies forward messages to agents when messages are specified in the policies.

– If an agent declares a *forward* policy for another agent, when specified messages are sent to other agents, the messages are forwarded to the latter as well as the former.
– If an agent declares a *delegate* policy for another agent, when specified messages are send to the former, the messages are forwarded to the latter but not to the former.

A *forward* policy is useful when two agents share the same information and *delegate* policy provides a master-slave relation between agents.

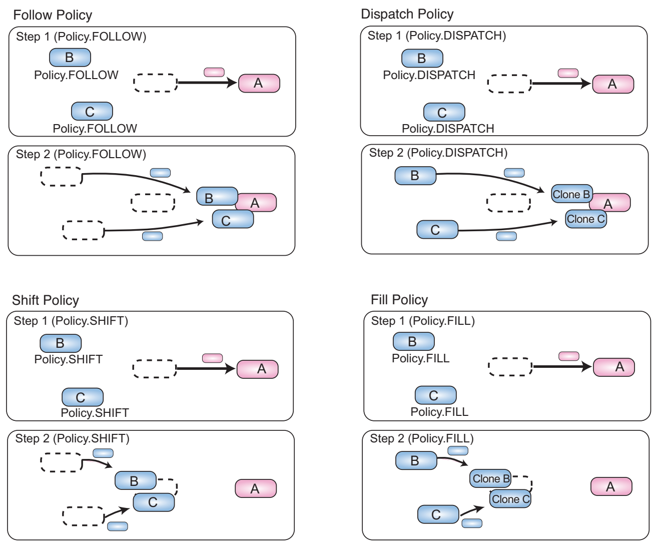

Fig. 2. Basic migration policies

3.2 Current Status

A prototype system of this framework was implemented with J2SE version 1.4.2 and although it was not built for performance, we measured the cost of agent migration.

The left of Figure 3 illustrates the cost of multiple-hops for two agents between two to eight computers (Pentium-M 1.4-MHz with Windows XP Professional and J2SE 1.4.2) through a fast ethernet, where the first agent declares a follow, dispatch, shift, or fill policy for the second and the second migrates between these computers sequentially without synchronizing the migration of the first.[1] Each cost in the left of Figure 3 is the latency of the first agent arriving after the second has begun to migrate to another computer. The cost of agent migration according to dispatch (or fill) policy is larger than the the follow (or shift) policy, because the former policy needs to create a copy of the first agent that has the policy. The cost of agent migration according to follow (or dispatch) is larger than that for dispatch (or shift), because the former and latter agents are deployed at different computers.

The right of Figure 3 shows the costs of multiple-hops of multiple agents between eight computers, when agents (from one to four) have follow, dispatch, shift, or fill policies for a moving agent. Unfortunately, with many hops is large, the follow and dispatch policies vary due to congestion at several computers. That is, two or more agents may attempt to have their own active threads in a single processor and to simultaneously transmit themselves to the destinations of their target agent in a TCP network connection. Once agents experience congestion at a computer, they tend to migrate as a chunk of agents rather than as individual agents to further destinations and the chunk often engulfs other newly arrival agents. The congestion does not always reappear, since computers are not synchronized and congestion often causes larger congestion in the routes of agents. We expect that fluctuations in the cost of agent migration will be large in a large-scale, heterogenous, distributed system.

4 Initial Experience

This section presents several examples that illustrate how the framework works.

4.1 Ants-based routing mechanisms

Ants are able to locate a path to a food source using trails of chemical substances called pheromones that are deposited by other ants. Several researchers have attempted to use the notion of ant pheromones for network-routing mechanisms [2, 11]. Our framework allows moving agents to leave themselves on their trails and to become automatically volatilized after their life-spans are over. A mobile agent corresponding to an ant, A, corresponding to a pheromone is attached to another mobile agent corresponding to an

[1] The latency between two computers is measured as the half-time of round-trip time between the source and destination computers. To measure latency between more than three computers exactly, these computers are connected through a ring topology. That is, the start and and goal of the second agent are assigned to the same computer and we measure difference between the timings of the first agent's starting and the second's arriving at the computer.

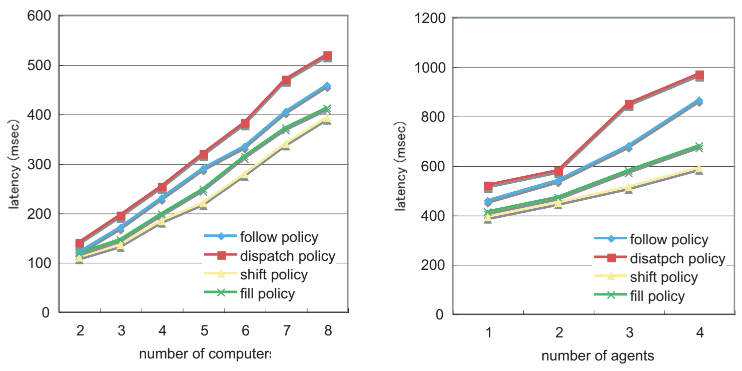

Fig. 3. Cost of multiple-hops for two agents between two to eight computers (left) and Costs of multiple-hops of multiple agents between eight computers (right).

ant according to the *fill* policy. When the latter agent randomly selects its destination and migrates to the selected destination, the former agent creates a clone and migrates to the source host of the latter. Since each of the cloned agents defines its life-span, they are active for a specified duration after being created. If there are other agents corresponding to pheromones in the host, the visiting agent adds their time spans to its own time span. When another agent corresponding to another ant migrates over the network, it can select a host that has the agents corresponding to pheromones whose time-spans are the longest from the neighboring hosts. We experimented on ant-based routing for mobile agents using this prototype implementation with more than eight computers. However, we knew that it would be difficult to quickly converge a short-path to the destination in real distributed systems, because routing mechanisms tend to be diverging.

4.2 Agent diffusion in sensor networks

The second example is the speculative deployment of agents as is done with cell-lamellipodia. This provides a mechanism that dynamically and speculatively deploys agents at sensor nodes when there are environmental changes. This mechanism was inspired by lamellipodia in cells. It assumes that the sensor field is a two-dimensional surface composed of sensor nodes and it monitors environmental changes, such as motion in objects and variations in temperature. It is a well known fact that after a sensor node detects environmental changes in its area of coverage, some of its geographically neighboring nodes tend to detect similar changes after a short time. Diffusion occurs as follows. When an agent on a sensor node finds changes in its environment, the agent duplicates itself and deploys the copy at neighboring nodes as long as the nodes have the same kinds of agents. Each agent is associated with a resource limit that functions as a generalized Time-To-Live (TTL) field. Although a node can monitor changes in interesting environments, it sets the TTLs of its agents as their own initial value. It otherwise decrements TTLs as the passage of time. When the TTL of an agent becomes zero, the agent automatically removes itself.

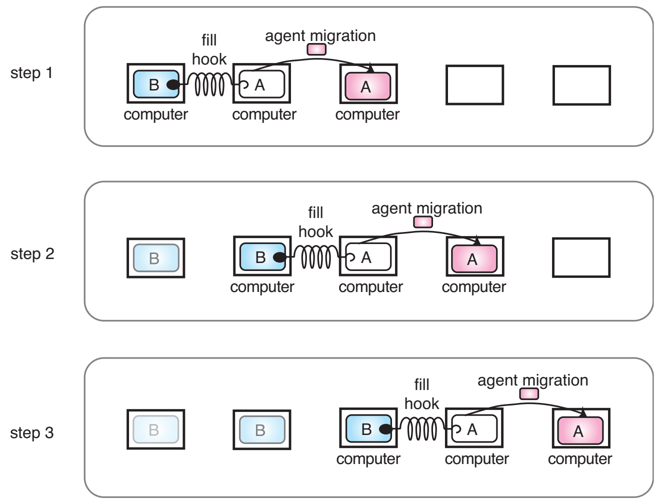

Fig. 4. Implementation of ant-based routing mechanism

5 Related Work

The section discusses several bio-inspired approaches to distributed and multi-agents systems. A few attempts have provided infrastructures for real distributed systems, like ours. The Anthill project [1] by the University of Bologna developed a bio-inspired middleware for peer-to-peer systems, which is composed of a collection of interconnected nests. Autonomous agents, called ants can travel across the network trying to satisfy user requests, like ours. The main difference between Anthill, including its applications, and our framework is that it introduces agents as independent entities and ours permits components to be organized in a self-organized manner. The Co-Field project [5] by the University di Modena e Reggio Emilia proposed the notion of a computational force-field model for coordinating the movements of a group of agents, including mobile devices, mobile robots, and sensors. However, the model only seems to be available within the limits of simulation and not within a real distributed system. Our deployment policies may be similar to the dynamic layout of distributed applications in the FarGo system [4]. However, FarGo's policies aim at allowing an agent to control other agents, whereas our policies aim at allowing an agent to describe its own migration, because our framework always treats agents as autonomous entities that travel from computer to computer under their own control. FarGo's policies may conflict when two agents can declare different relocation policies for a single agent. However, our framework is free of any conflict because each agent can only declare a policy to relocate itself instead of other agents. The author presented a bio-inspired deployment of software components [10]. The previous approach is an early implementation of the framework presented in this paper. It supported some of the deployment policies but not any message policies.

6 Conclusion

This paper presented a middleware system for dynamically deploying agents at different computers, instead of any simulation-based systems. We designed and implemented a prototype system of them middleware and demonstrated its effectiveness in several applications. Since the middleware enabled each agent to specify its own policy as a relocation between the agent and another agent, it cannot only move individual agents but also a federation of agents over a distributed system in a self-organized manner.

We would like to point out further issues that need to be resolved. We need various evaluations on real distributed systems. Although the current implementation focuses on the deployment of agents, we plan to extend it so that it can be used to modify the behavior of each agent, while they are running.

References

1. O. Babaoglu and H. Meling and A. Montresor, Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems, Proceeding of 22th IEEE International Conference on Distributed Computing Systems, July 2002.
2. G. Di Caro and M. Dorigo, AntNet: Distributed Stigmergetic Control for Communications Networks, Journal of Artificial Intelligence Research, vol.9, pp. 317-365, 1998.
3. B. Horling, and V. Lesser, and R. Vincent, Multi-Agent System Simulation Framework Proceeding of IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, August 2000.
4. O. Holder, I. Ben-Shaul, and H. Gazit, System Support for Dynamic Layout of Distributed Applications, Proceedings of International Conference on Distributed Computing Systems (ICDCS'99), pp 403-411, IEEE Computer Society, 1999.
5. M. Mamei, L. Leonardi, F. Zambonelli, Co-Fields: A Unifying Approach to Swarm Intelligence, International Workshop on Engineering Societies in the Agents World (ESAW 2002), Lecture Notes in Computer Science, vol. 2577, Springer Verlag 2003.
6. N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System, A Toolkit for Building Multi-Agent Simulations, Technical report, Swarm Development Group, June 1996.
7. I. Satoh, Building Reusable Mobile Agents for Network Management, IEEE Transactions on Systems, Man and Cybernetics, vol.33, no. 3, part-C, pp.350-357, August 2003.
8. I. Satoh, Configurable Network Processing for Mobile Agents on the Internet Cluster Computing (The Journal of Networks, Software Tools and Applications), vol. 7, no.1, pp.73-83, Kluwer, January 2004.
9. I. Satoh, Selection of Mobile Agents, Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS'2004), pp.484-493, IEEE Computer Society, March 2004.
10. I. Satoh, Bio-inspired Deployment of Distributed Applications, Proceedings of International Workshop on Multi-Agents (PRIMA2004), Lecture Notes in Computer Science (LNCS), vol.3371,pp.243-258, Springer, August 2004.
11. R. Schoonderwoerd, O. Holland, and J. Bruten, Ant-like agents for load balancing in telecommunications networks, Proceedings of Conference on Autonomous Agents, pages 209-216. ACM Press, 1997.
12. World Wide Web Consortium (W3C), Composite Capability/Preference Profiles (CC/PP), http://www.w3.org/TR/NOTE-CCPP, 1999.