Scalable Agent Distribution Mechanisms for Large-Scale UAV Simulations

Myeong-Wuk Jang and Gul Agha
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{mjang, agha}@uiuc.edu

Abstract *— A cluster of computers is required to execute large-scale multi-agent. However, such execution incurs an inter-node communication overhead because agents intensively communicate with other agents to achieve common goals. Although a number of dynamic load balancing mechanisms have been developed, these mechanisms are not scalable in multi-agent applications because of the overhead involved in analyzing the communication patterns of agents. This paper proposes two scalable dynamic agent distribution mechanisms; one mechanism aims at minimizing agent communication cost, and the other mechanism attempts to move agents from overloaded agent platforms to lightly loaded platforms. Our mechanisms are fully distributed algorithms and analyze only coarse-grain communication dependencies of agents, thus providing scalability. We describe the results of applying these mechanisms to large-scale micro UAV (Unmanned Aerial Vehicle) simulations involving up to 10,000 agents.*

1. INTRODUCTION

As the number of agents in large-scale multi-agent applications increases by orders of magnitude (e.g. see [7, 9, 10]), distributed execution is required to improve the overall performance of applications. However, parallelizing the execution on a cluster may lead to inefficiency; a few computer nodes may be idle while others are overloaded. Many dynamic load balancing mechanisms have been developed to enable efficient parallelization [1, 3, 6]. However, these mechanisms may not be applicable to multi-agent applications because of the different computation and communication behavior of agents [4, 9].

Some load balancing mechanisms have been developed for multi-agent applications [4, 5], but these mechanisms require a significant overhead to gather information about the communication patterns of agents and analyze the information. Therefore, we believe these mechanisms may not be scalable. In this paper, we propose two *scalable* agent distribution mechanisms; one mechanism aims at minimizing agent communication cost, and the other mechanism attempts to move agents from an overloaded agent platform to lightly loaded agent platforms. These two mechanisms are developed as fully distributed algorithms and analyze only coarse-grain communication dependencies of agents instead of their fine-grain communication dependencies.

Although the scalability of multi-agent systems is an important concern in the design and implementation of multi-agent platforms, we believe such scalability cannot be achieved without customizing agent platforms for a specific multi-agent application. In our agent systems, each computer node has one agent platform, which manages scheduling, communication, and other middleware services for agents executing on the computer node. Our multi-agent platform is adaptive to improve the scalability of the entire system. Specifically, large-scale micro UAV (Unmanned Aerial Vehicle) simulations involving up to 10,000 agents are studied using our agent distribution mechanisms.

The paper is organized as follows: Section 2 discusses the scalability issues of multi-agent systems. Section 3 describes two agent distribution mechanisms implemented in our agent platform. Section 4 explains our UAV simulations and their interaction with our agent distribution mechanisms. Section 5 shows the preliminary experimental results to evaluate the performance gain resulting from the use of these mechanisms. The last section concludes this paper with a discussion of our future work.

2. SCALABILITY OF MULTI-AGENT SYSTEMS

The scalability of multi-agent systems depends on the structure of an agent application as well as the multi-agent platform. For example, when a distributed multi-agent application includes centralized components, these components can become a bottleneck of parallel execution, and the application may not be scalable. Even when an application has no centralized components, agents may use middle agent services, such as brokering or matchmaking services, supported by agent platforms, and the agent platform-level component that supports these services may

KIMAS 2005, April 18-21, 2005, Waltham, MA, USA.
0-7803-9013-X/05/$20.00 © 2005 IEEE.

become a bottleneck for the entire system.

The goal of executing a cluster of computers for a single multi-agent application is to improve performance by taking advantage of parallel execution. However, balancing the workload on computer nodes requires a significant overhead from gathering the global state information, analyzing the information, and transferring agents very often among computer nodes. When the number of computer nodes and/or that of agents are very large, achieving optimal load balancing is not feasible. Therefore, we use a load sharing approach which move agents from an overloaded computer node, but the workload balance between different computer nodes is not required to be optimal.

Another important factor in the performance of large-scale multi-agent applications is agent communication cost. This cost may significantly affect the performance of multi-agent systems, when agents distributed on separate computer nodes communicate intensively with each other. Even though the speed of local networks has considerably increased, the intra-node communication for message passing is much faster than inter-node communication. Therefore, if we can collocate agents which communicate intensively with each other, communication time may significantly decrease. Distributing agents statically by a programmer is not generally feasible, because the communication patterns among agents may change over time. Thus agents should be dynamically reallocated according to their communication localities of agents, and this procedure should be managed by a multi-agent platform.

Because of a large number of agents in a single multi-agent application, the overhead from gathering the communication patterns of agents and analyzing such patterns would significantly affect the overall performance of the entire system. For example, when there are n agents and unidirectional communication channels between agents are used, the maximum number of possible communication connections among agents is $n \times (n\text{-}1)$. If the communication patterns between agents and agent platforms are considered for dynamic agent distribution, the maximum number of communication connections becomes $n \times m$ where m is the number of agent platforms. Usually, m is much less than n.

Another important concern for the scalability is the location of agent distributor that performs dynamic agent distribution. If a centralized component handles this task, the communication between this component and agent platforms may be significantly increased and the component may be the bottleneck of the entire system. Therefore, when multi-agent systems are large-scale, more simplified information for decision making and distributed algorithms would be more applicable for the scalability of dynamic agent distribution mechanisms.

For the purpose of dynamic agent distribution, each agent platform may monitor the status of its computer node and the communication patterns of agents on it, and distribute agents according to their communication localities and the workload of its computer node. However, with the interaction with multi-agent applications, the quality of this service may be improved. For example, multi-agent applications may initialize or change parameters of dynamic agent distribution during execution for the better performance of the entire system. For the interaction between agent applications and platforms, we use a *reflective mechanism* [11]; agents in applications are supported by agent platforms, and the services of agent platforms may be controlled by agents in applications. This paper shows how our multi-agent applications (e.g., UAV simulations) interact with our dynamic agent distribution mechanisms.

3. DYNAMIC AGENT DISTRIBUTION

This section describes two mechanisms for dynamic agent distribution: a *communication localization* mechanism collocates agents which communicate intensively with each other, and a *load sharing* mechanism moves agent groups from overloaded agent platforms to lightly loaded agent platforms. Although the purpose of these two mechanisms is different, the mechanisms consist of similar process phases and share the same components in an agent platform. Both these two mechanisms are also designed as fully distributed algorithms. Figure 1 shows the state transition diagram for these two agent distribution mechanisms.

For these dynamic agent distribution services, four system components in our agent platform are mainly used. A detailed explanation of system components is described in [9].

1. *Message Manager* takes charge of message passing between agents.

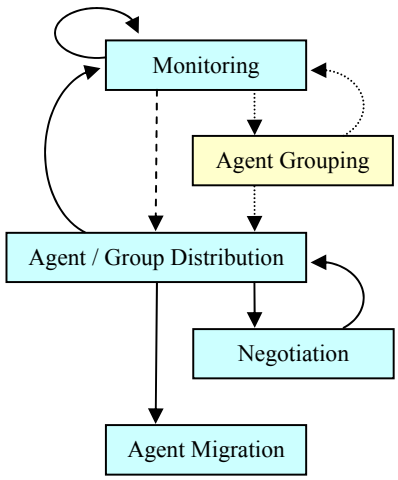

Figure 1 - State Transition Diagram for Dynamic Agent Distribution. The solid lines are used for both mechanisms, the dashed line is used only for the communication localization mechanism, and the dotted lines are used only for the load sharing mechanism.

2. *System Monitor* periodically checks the workload of its computer node.
3. *Actor Allocation Manager* is responsible for dynamic agent distribution.
4. *Actor Migration Manager* moves agents to other agent platforms.

3.1. Agent Distribution for Communication Locality

The communication localization mechanism handles the dynamic change of the communication patterns of agents. As time passes, the communication localities of agents may change according to the changes of agents' interests. By analyzing messages delivered between agents, agent platforms may decide what agent platform an agent should be located on. Because an agent platform can neither estimate the future communication patterns of agents nor know how agents on other platforms may migrate, local decision of an agent platform cannot be perfect. However, our experiments show that in case of our applications, reasonable performance can be achieved. The communication localization mechanism consists of four phases: *monitoring*, *agent distribution*, *negotiation*, and *agent migration* (see Figure 1).

Monitoring Phase — The Actor Allocation Manager checks the communication patterns of agents with the assistance from the Message Manager. Specifically, the Actor Allocation Manager uses information about both the sender agent and the agent platform of the receiver agent of each message. This information is maintained with a variable M representing all agent platforms communicating with each agent on the Manager's platform.

The Actor Allocation Manager periodically computes the communication dependencies $C_{ij}(t)$ at time t between agent i and agent platform j using equation 1.

$$C_{ij}(t) = \alpha\left(\frac{M_{ij}(t)}{\sum_k M_{ik}(t)}\right) + (1-\alpha)C_{ij}(t-1) \qquad (1)$$

where $M_{ij}(t)$ is the number of messages sent from agent i to agent platform j during the t-th time step, and α is a coefficient representing the relative importance between recent information and old information.

Agent Distribution Phase — After a certain number of repeated monitoring phases, the Actor Allocation Manager computes the communication dependency ratio of an agent between its current agent platform n and all other agent platforms, where the communication dependency ratio R_{ij} between agent i and platform j is defined using equation 2.

$$R_{ij} = \frac{C_{ij}}{C_{in}}, \quad j \neq n \qquad (2)$$

When the maximum value of the communication dependency ratio of an agent is larger than a predefined threshold θ, the Actor Allocation Manager assigns the agent to a *virtual agent group* that represents a remote agent platform.

$$k = \arg\max_j(R_{ij}) \quad and \quad R_{ik} > \theta \quad \rightarrow \quad a_i \in G_k \qquad (3)$$

where a_i represents agent i, and G_k means virtual agent group k.

After the Actor Allocation Manager checks all agents, and assigns some of them to virtual agent groups, it starts the negotiation phase, and information about the communication dependencies of agents is reset.

Negotiation Phase — Before the agent platform P_1 moves the agents assigned to a given virtual agent group to destination agent platform P_2, the Actor Allocation Manager of P_1 communicates with that of P_2 to check the current status of P_2. Only if P_2 has enough space and the percentage of its CPU usage is not continuously high, the Actor Allocation Manager of P_2 accepts the request. Otherwise, the Manager of P_2 responds with the number of agents that it can accept. In this case, the P_1 moves only a subset of the virtual agent group.

Agent Migration Phase — Based on the response of a destination agent platform, the Actor Allocation Manager of the sender agent platform initiates migration of entire or part of agents in the selected virtual agent group. When the destination agent platform has accepted part of agents in the virtual agent group, agents to be moved are selected according to their communication dependency ratios. After the current operation of a selected agent finishes, the Actor Migration Manager moves the agent to its destination agent platform. After the agent is migrated, it carries out its remaining operations.

3.2. Agent Distribution for Load Sharing

The agent distribution mechanism for communication locality handles the dynamic change of the communication patterns of agents, but this mechanism may overload a platform once more agents are added to this platform. Therefore, we provide a load sharing mechanism to redistribute agents from overloaded agent platforms to lightly loaded agent platforms. When an agent platform is overloaded, the System Monitor detects this condition and activates the agent redistribution procedure. Since agents had been assigned to their current agent platforms according to their recent communication localities, choosing agents randomly for migration to lightly loaded agent platforms may result in cyclical migration. The moved agents may still have high communication rate with agents on their previous agent platform. Our load sharing mechanism consists of five phases: *monitoring*, *agent grouping*, *group distribution*, *negotiation*, and *agent migration* (see Figure 1).

Monitoring Phase — The System Monitor periodically

checks the state of its agent platform; the System Monitor gets information about the current processor usage and the memory usage of its computer node by accessing system call functions and maintains the number of agents on its agent platform. When the System Monitor decides that its agent platform is overloaded, it activates an agent distribution procedure. When the Actor Allocation Manager is notified by the System Monitor, it starts monitoring the local communication patterns of agents in order to partition them into *local agent groups*. For this purpose, an agent which was not previously assigned to an agent group is randomly assigned to some agent group.

To check the local communication patterns of agents, the Actor Allocation Manager uses information about the sender agent, the agent platform of the receiver agent, and the agent group of the receiver agent of each message. After a predetermined time interval the Actor Allocation Manager updates the communication dependencies between agents and local agent groups on the same agent platform using equation 1 using $c_{ij}(t)$ and $m_{ij}(t)$ instead of $C_{ij}(t)$ and $M_{ij}(t)$. In the modified equation 1, j represents a local agent group instate of an agent platform, $c_{ij}(t)$ represents the communication dependency between agent i and agent group j at the time step t, and $m_{ij}(t)$ is the number of messages sent from agent i to agents in local agent group j during the t-th time step. Note that in this case $\sum_k m_{ik}(t)$ represents the number of messages sent by the agent i to all agents in its current agent platform, and in general the value of the parameter α will be different.

Agent Grouping Phase — After a certain number of repeated monitoring phases, each agent i is re-assigned to a local agent group whose index is decided by $\arg\max_j(c_{ij}(t))$. Since the initial group assignment of agents may not be well organized, the monitoring and agent grouping phases are repeated several times. After each agent grouping phase, information about the communication dependencies of agents is reset.

Group Distribution Phase — After a certain number of repeated monitoring and agent grouping phases, the Actor Allocation Manager makes a decision to move an agent group to another agent platform. The group selection is based on the communication dependencies between agent groups and agent platforms. Specifically the communication dependency D_{ij} between local agent group i and agent platform j is decided by summing the communication dependencies between all agents in the local agent group and the agent platform. Let A_i be the set of indexes of all agents in the agent group i.

$$D_{ij} = \sum_{k \in A_i} C_{kj}(t) \tag{4}$$

where $C_{kj}(t)$ is the communication dependency between agent k and agent platform j at time t.

The agent group which has the least dependency to the current agent platform is selected using equation 5.

$$\arg\max_i \left(\frac{\sum_{j, j \neq n} D_{ij}}{D_{in}} \right) \tag{5}$$

where n is the index of the current agent platform.

The destination agent platform of the selected agent group i is decided by the communication dependency between the agent group and agent platforms using equation 6.

$$\arg\max_j(D_{ij}) \quad where \quad j \neq n \tag{6}$$

Negotiation Phase — If one agent group and its destination agent platform are decided, the Actor Allocation Manager communicates with that of the destination agent platform. If the destination agent platform accepts all agents in the group, the Actor Allocation Manager of the sender agent platform starts the migration phase. Otherwise, this Actor Allocation Manager communicates with that of the second best destination platform until it finds an available destination agent platform or checks the feasibility of all other agent platforms.

This phase of our load sharing mechanism is similar to that of the communication localization mechanism. However, the granularity of negotiation for these two mechanisms is different: the communication localization mechanism is at the level of an agent while the load sharing mechanism is at the level of an agent group. If the destination agent platform has enough space and available computation resource for all agents in the selected local agent group, the Actor Allocation Manager of the destination agent platform can accept the request for the agent group migration. Otherwise, the destination agent platform refuses the request; the destination agent platform cannot accept part of a local agent group.

Agent Migration Phase — When the sender agent platform receives the acceptance reply from the receiver agent platform, the Actor Allocation Manager of the sender agent platform initiates migration of entire agents in the selected local agent group. The following procedure for this phase in the agent distribution mechanism for load sharing is the same as that in the agent distribution mechanism for communication locality.

4. UAV SIMULATIONS

Our dynamic agent distribution mechanisms have been applied to large-scale UAV (Unmanned Aerial Vehicle) simulations. The purpose of these simulations is to analyze the cooperative behavior of micro UAVs under given situations. Several coordination schemes have been simulated and compared to the performance of a selfish UAV scheme. These UAV simulations are based on the

agent-environment interaction model [2]; all UAVs and targets are implemented as intelligent agents, and the navigation space and radar censors of all UAVs are simulated by environment agents. To remove centralized components in distributed computing, each environment agent on a single computer node is responsible for a certain navigation area. In addition to direct communication of UAVs with their names, UAVs may communicate indirectly with other agents through environment agents without agent names. Environment agents use the *ATSpace* model to provide application agent-oriented brokering services [8]. During simulation time, UAVs and targets move from one divided area to another, and they communicate intensively either directly or indirectly.

Figure 2 depicts the components for our UAV simulations. These simulations consist of two types of agents: task-oriented agents and simulation-oriented agents. Task-oriented agents simulate objects in a real situation. For example, a UAV agent represents a physical micro UAV, while a target represents an injured civilian or solider to be searched and rescued. For simulations, we also need simulation-oriented agents: *Simulation Control Manager* and *Environment Simulator*. Simulation Control Manager synchronizes local virtual times of components, while Environment Simulator simulates both the navigation space and the local broadcasting and radar sensing behavior of all UAVs.

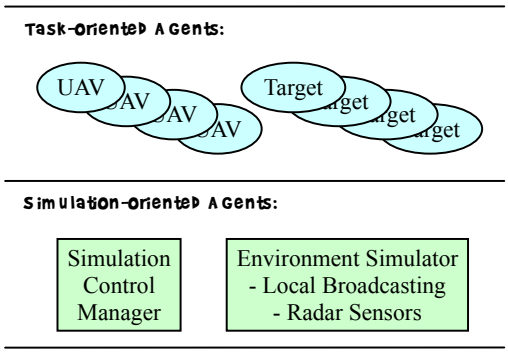

Figure 2 – Architecture of UAV Simulator

When a simulation starts, Simulation Control Manager initializes the parameters of dynamic agent distribution. These parameters include the coefficients for two types of communication dependencies (i.e. agent platform level and agent group level), the migration threshold, the number of local agent groups, and the relative frequency of monitoring and agent grouping phases. During execution, the size of each time step t in dynamic agent distribution is also controlled by Simulation Control Manager; this size is the same as the size of a simulation time step. Thus, the size of time steps varies according to the workload of each simulation step and the processor power. Moreover, the parameters of dynamic agent distribution may be changed by Simulation Control Manager during execution.

5. Experimental Results

We have conducted experiments with micro UAV simulations. These simulations include from 2,000 to 10,000 agents; half of them are UAVs, and the others are targets. Micro UAVs perform a surveillance mission on a mission area to detect and serve moving targets. During the mission time, these UAVs communicate with their neighboring UAVs to perform the mission together. The size of a simulation time step is one half second, and the total simulation time is around 37 minutes. The runtime of each simulation depends on the number of agents and the selected agent distribution mechanism. For these experiments, we use four computers (3.4 GHz Intel CPU and 2 GB main memory) connected by a Giga-bit switch.

Figure 3 and Figure 4 depict the performance benefit of dynamic agent distribution in our experiments comparing with static agent distribution. Even though the dynamic agent distribution mechanisms in our simulations include the overhead from monitoring and decision making, the overall performance of simulations with dynamic agent distribution is much better than that with static agent

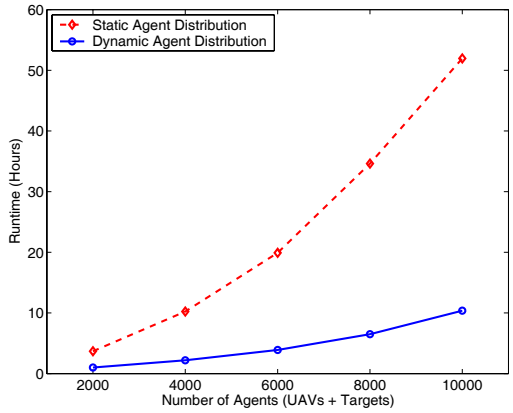

Figure 3 - Runtime of Simulations using Static and Dynamic Agent Distribution

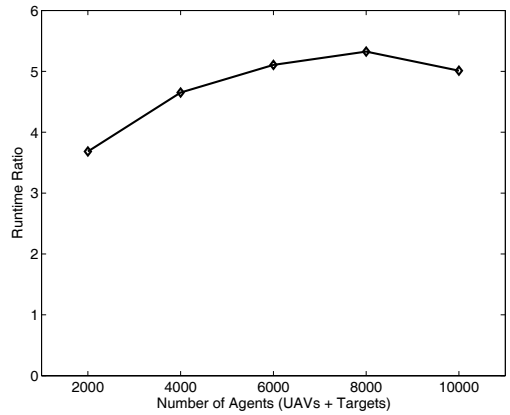

Figure 4 - Runtime Ratio of Static-to-Dynamic Agent Distribution

distribution. In our particular example, as the number of agents is increased, the ratio also generally increases. The simulations using dynamic agent distribution is more than five times faster than those using static agent distribution when the number of agents is large.

6. CONCLUSION

This paper has explained two scalable agent distribution mechanisms used for UAV simulations; these mechanisms distribute agents according to their communication localities and the workload of computer nodes. The main contributions of this research are that our agent distribution mechanisms are based on the dynamic changes of communication localities of agents, that these mechanisms focus on the communication dependencies between agents and agent platforms and the dependencies between agent groups and agent platforms, instead of the communication dependencies among individual agents, and that our mechanisms continuously interact with agent applications to adapt dynamic behaviors of an agent application. In addition, these agent distribution mechanisms are fully distributed mechanisms, are transparent to agent applications, and are concurrently executed with them.

The proposed mechanisms introduce an additional overhead for monitoring and decision making for agent distribution. However, our experiments suggest that this overhead are more than compensated when multi-agent applications have the following attributes: first, an application includes a large number of agents so that the performance on a single computer node is not acceptable; second, some agents communicate more intensively with each other than with other agents, and thus the communication locality of each agent is an important factor in the overall performance of the application; third, the communication patterns of agents are continuously changing, and hence, static agent distribution mechanisms are not sufficient.

In our multi-agent systems, the UAV simulator modifies the parameters of dynamic agent distribution to improve its quality. However, some parameters are currently given by application developers, and finding these values requires developers' skill. We plan to develop learning algorithms to automatically adjust these values during execution of applications.

ACKNOWLEDGEMENTS

The authors would like to thank Sandeep Uttamchandani for his helpful comments and suggestions. This research is sponsored by the Defense Advanced Research Projects Agency under contract number F30602-00-2-0586.

REFERENCES

[1] K. Barker, A. Chernikov, N. Chrisochoides, and K. Pingali, "A Load Balancing Framework for Adaptive and Asynchronous Applications," *IEEE Transactions on Parallel and Distributed Systems*, 15(2):183-192, February 2004.

[2] M. Bouzid, V. Chevrier, S. Vialle, and F. Charpillet, "Parallel Simulation of a Stochastic Agent/Environment Interaction," *Integrated Computer-Aided Engineering*, 8(3):189-203, 2001.

[3] R.K. Brunner and L.V. Kalé, "Adaptive to Load on Workstation Clusters," *The Seventh Symposium on the Frontiers of Massively Parallel Computations*, pages 106-112, February 1999.

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

[5] T. Desell, K. El Maghraoui, and C. Varela, "Load Balancing of Autonomous Actors over Dynamics Networks," *Hawaii International Conference on System Sciences HICSS-37 Software Technology Track*, Hawaii, January 2004.

[6] K. Devine, B. Hendrickson, E. Boman, M.St. John, C. Vaughan, "Design of Dynamic Load-Balancing Tools for Parallel Applications," *Proceedings of the International Conference on Supercom0puting*, pages 110-118, Santa Fe, 200l

[7] L. Gasser and K. Kakugawa, "MACE3J: Fast Flexible Distributed Simulation of Large, Large-Grain Multi-Agent Systems," *Proceedings of the First International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pages 745-752, Bologna, Italy, July 2002.

[8] M. Jang, A. Abdel Momen, and G. Agha, "ATSpace: A Middle Agent to Support Application-Oriented Matchmaking and Brokering Services," *IEEE/WIC/ACM IAT(Intelligent Agent Technology)-2004*, pages 393-396, Beijing, China, September 2004.

[9] M. Jang and G. Agha, Dynamic Agent Allocation for Large-Scale Multi-Agent Applications, *Proceedings of International Workshop on Massively Multi-Agent Systems*, Kyoto, Japan, December 2004.

[10] K. Popov, V. Vlassov, M. Rafea, F. Holmgren, P. Brand, and S. Haridi, "Parallel Agent-based Simulation on Cluster of Workstations," *Parallel Processing Letters*, 13(4):629-641, 2003.

[11] D. Sturman, *Modular Specification of Interaction Policies in Distributed Computing*, PhD thesis, University of Illinois at Urbana-Champaign, May 1996.

[12] C. Walshaw, M. Cross, and M. Everett, "Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes," *Journal of Parallel and Distributed Computing*, 47:102-108, 1997.