Agent cloning: an approach to agent mobility and resource allocation *

Onn Shehory, Katia Sycara, Prasad Chalasani, and Somesh Jha
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
onn,katia,chal,sjha@cs.cmu.edu

Abstract

Multi-agent systems (MAS) are subject to performance bottlenecks in cases where agents cannot perform tasks by themselves due to insufficient resources. Solutions to such problems include passing tasks to others or agent migration to remote hosts. We propose agent cloning as a more comprehensive approach to the problem of local agent overloads. Agent cloning subsumes task transfer and agent mobility. According to our paradigm, agents may clone, pass tasks to others, die or merge. We discuss the requirements of implementing a cloning mechanism and its benefits in a Multi-Agent System, and support our claims with simulation results.

*This material is based upon work supported in part by ARPA Grant #F33615-93-1-1330, by ONR Grant #N00014-96-1222, and by NSF Grant #IRI-9612131.

1 Introduction

Multi-agent systems (MAS) provide efficient solutions for several computational problems. However, as in other computational solutions, one will expect limitations and bottlenecks. These may stem either from the components of the computational environment in which the agents are situated (e.g., malfunctioning communication lines), or from the intrinsic multi-agent system characteristics and tasks. System researchers have developed a variety of methods to overcome faults and bottlenecks to increase the efficiency of systems. In multi-agent systems, the distributed nature of the system and the autonomy of the agents may contribute to the ability to overcome overloading bottlenecks. This paper presents such a solution.

We have a multi-agent system that receives a stream of tasks. In our framework, an agent is an autonomous, self-aware, intelligent and pro-active computational entity. An agent is providing services by performing tasks which it either generates by itself or are delegated to it by users or other agents. A task is either an executable code or a goal represented in a higher level of abstraction. In the latter case, an agent should be able to transform the abstract goal into more concrete tasks. The concrete tasks are examined by the agent to verify whether their performance is within its capabilities. If they are, the agent either performs the tasks as given or further decomposes to sub-tasks. Otherwise the tasks may be delegated to the appropriate agents. A MAS is a group of agent as described above. The members of the system may each receive tasks and may perform or delegate the task for performance by others. The agents have *capabilities* which indicate the types of tasks they can perform and *capacities* which indicate the amounts of resources that the agents can access and use for task execution. Tasks are categorized by types that can be handled by agents with appropriate capabilities. The problem discussed in this paper is the situation where the task flow to an agent overloads it. Such overloads are of two different general categories:

1. An agent in a MAS is overloaded, but the MAS as a whole has the required capabilities and capacities.

2. The MAS as a whole is overloaded, i.e., the agents that comprise the MAS do not have the necessary capacities (however there may be idle resources in the computational system where the agents are situated).

As a result of such overloads, the MAS will not perform all of the tasks in time, although the required resources may be available to it. The following solutions suggest themselves:

1. First case – overloaded agents should pass tasks to others agents which have the capabilities and capacities to perform them.

2. Second case – overloaded agents create new agents to perform excess tasks and utilize unused resources or migrate to other hosts.

In this paper we present *agent cloning* as a means for implementing these solutions and discuss the required reasoning, decision making and actions that are necessary for an agent within the system to perform cloning. Agent migration can be implemented by creating a clone on a remote machine, transferring the tasks from the original agent to the clone, and dying. Thus, agent mobility is an instance of agent cloning.

In particular, we consider cloning in an open environment, such as the Internet, where agents might dynamically and unpredictably appear or disappear. To study MAS issues in such an environment, we use the RETSINA domain-independent infrastructure [13]. In RETSINA, there are three types of agents: *providers*, who possess certain capabilities and perform tasks that require these capabilities; *requesters*, who have tasks to be performed and who locate agents with the required capabilities to whom they delegate the tasks; *middle agents* such as matchmakers [5], by whom requester agents locate provider agents. Provider agents advertise their capabilities to middle agents and requesters ask the latter to find providers with required capabilities. When an agent perceives an overload, the RETSINA infrastructure allows it to find (through middle agents) providers with appropriate capabilities (similar to its own) to whom it can consider transferring tasks. When no such providers are found, a clone may be created. The clone, like any other agent in the society, advertises itself with a middle agent. Thus, the clone becomes known to the multi-agent society.

In this paper we provide an analysis of the circumstances under which agents should consider cloning, and present experimental results on how cloning affects the performance of a MAS. Our simulation results show that using our cloning protocols to address local overloading problems, improves agent and system performance.

2 The cloning approach

Cloning is a possible response of an agent to overloads. Agent overloads are due, in general, either to the agent's limited capacity to process current tasks or to machine overloads. Other approaches to overloads include task transfer and agent migration. Task transfer, where overloaded agents locate other agents which are lightly loaded and transfer tasks to them, is very similar to processor load balancing. Agent migration, which requires that overloaded agents, or agents that run on an overloaded machine (these loads are different but may correlate), migrate to less loaded machines, is closely related to process migration and to the recently emerging field of mobile agents [1]. A main difference between load balancing and agent cloning is that, while the first explicitly discusses machine loads and agent migration, the latter, in addition, considers a different type of load – the agent load.

Therefore, cloning is a superset of task transfer and agent migration: it includes them and adds to them as well. Cloning does not necessarily require migration to other machines. Rather, a new agent is created either on the local or a remote machine. Note that there may be several agents running on the same machine, and having one of them overloaded does not necessarily imply that the others are overloaded (although we expect some correlation between overloads). Agent overload does not imply machine overload, and therefore local cloning (that is, on the same machine) may be possible. As shown in the load balancing literature [8], within a distributed system there is a high probability of having some of the processors idle, while others are highly loaded. Cloning takes advantage of these idle processing capacities.

To perform cloning, an agent must reason about its own load (current and future), its host load as well as capabilities and loads of other machines and agents. Accordingly, it may decide to: create a clone; pass tasks to a clone; merge with other agents; or die. Merging of two agents, or self-extinction of underutilized agents is an important mechanism to control agent proliferation with resulting overload of network resources. Detailed consideration of this problem, however, is outside of the scope of this paper.

To avoid communication overhead in trying to access and reason about remote hosts, reasoning regarding cloning begins by considering local cloning. When this is found infeasible or non-beneficial, the agent proceeds to reason about remote cloning. If remote cloning is decided upon, an agent should be created and activated on a remote machine. Assuming that the agent has an access and a permit to work on this machine, there may be two main methods to perform this cloning: (1) creating the agent locally and letting it migrate to the remote machine (similar to a mobile agent) or, (2) creating and activating the agent on the remote machine. While the first method requires very little on the part of the remote machine, it requires mobilization properties as well as additional local resource consumption. The second method, while avoiding mobilization and local resource consumption, requires that a copy of the agents' code be located on the remote machine. Similar requirements also hold for mobile agent applications [7, 14], since an agent server or an agent dock is required. Nonetheless, the amount of this code is small.

Since the agent's own load and the loads of other agents vary over time in a non-deterministic way, the decision of *whether and when* to clone is non-trivial. Prior work [4] has presented a model of cloning based on prediction of missed task deadlines and idle times on the agent's schedule in the RETSINA multi-agent infrastructure [13]. In this paper (section 3.1), we present a stochastic model of decision making based on dynamic programming to determine the optimal timing for cloning.

Suppose a clone has been created and activated. Several questions yet remain with respect to this clone. These regard its autonomy, its tasks, its lifetime, and its access to resources. *Autonomy* refers to independent clone vs. a subordinate one. Having been created and activated, an independent clone is not controlled by its creator. Therefore, such a clone will continue to exist after completion of the tasks provided by its initiator agent. Hence, a mechanism for deciding what it should do afterwards is necessary. Such a mechanism must allow the clone to reason about the agent- and task-environment, and accordingly decide whether it should continue to work on other tasks (if necessary and if the computational resources allow), merge with others, or perform self extinction.

A *subordinate* clone will remain under the control of its initiator. This will prevent the complications arising in the independent clone case - i.e., it is not necessary to decide what to do after the tasks that were delegated to the clone are accomplished. However, in order to manage a subordinate agent, the initiating agent must be provided with a control mechanism for remote agents. Regardless of the details of such a mechanism, it will require additional communication between the two agents, thus increasing the communication overhead of such a cloning method and the MAS vulnerability to communication flaws. In addition, control of other agents is a partially-centralized solution, which might violate the reason for using a multi-agent system in the first place.

3 Cloning Initiation

An agent should consider cloning if

- It cannot perform all of its tasks on time by itself nor can it decompose them so that they can be delegated to others, and
- There is no lightly-loaded agent that can receive and perform its excess tasks (or sub-tasks when tasks are decomposable), and
- There are sufficient resources for creating and activating a clone agent (either on the same machine or on a remote one), and
- The efficiency of the clone agent and the original agent is expected to be greater than that of the original agent alone.

The above requirements may be difficult for an agent to perceive. Later in this paper we discuss the methods according to which agents reason about themselves and their environment to achieve this perception.

The necessary information used by an agent to decide whether and when to initiate cloning comprises parameters that describe both local and remote resources. In particular, the necessary parameters are as follows:

- The CPU and memory loads, both internal to the agent (which result from planning, scheduling and task-execution activities of the agent) and external (on the agent host and possibly on remote hosts).
- The CPU execution speed (measured using standard methods e.g., MIPS), both locally and remotely.
- The load on the communication channels and their transfer rate, both locally and remotely.
- The current queue of tasks, the resources required for their execution and their deadlines.
- The future expected flow of tasks.

To acquire the above information an agent must be able to read the operating system variables. In addition, the agent must have self awareness on two levels – an agent internal level and a MAS level. The internal self awareness should allow the agent to realize what part of the operating system retrieved values are its own properties (that is, agent internal parameters). The system-wise self awareness should allow the agent to find, possibly via middle agents [5], information regarding available resources on remote machines. Without middle agents (e.g. matchmakers), servers that are located on the remote hosts can supply such information upon request. In our simulation we have examined both of these methods for acquiring information with regards to remote hosts and found no significant difference in the performance. When such information is not available, an agent may compute the expectation values of the attributes of remote machines relying on probability distributions either specifically by machine i.d. (e.g. IP address) or groupwise, by machine type.

3.1 Optimizing when to clone

To maximize the benefits of cloning, an agent should decide on performing it at the optimal time. Below, we provide a method for optimizing the cloning decision making. For this, each decision regarding cloning has a value, calculated with respect to loads and distances as a function of time. Here, loads refer to processing load, memory load and communication load. The distance between agents is the cumulative distance according to the communication route between them. An agent A_i has a valuation function $\mathrm{Val}_i(loads, distances)$, where $loads$ is a set of loads of agents and $distances$ is a set of distances to other agents. When measuring the time in discrete units, we describe the possible decisions of A_i by a decision tree. The tree includes a set of decision points which are the nodes of the tree. The edges of the tree are decisions and each is attached a value. These values may be discounted over time by a given discount rate r_i. The discount rate is used in cases where the agent assumes that the value of a decision is discounted over time (otherwise $r_i = 0$). We define a recursive function to evaluate decision making with dependency:

$$Value(v) = \begin{cases} 0 & \text{no decision taken} \\ Val(v) & \text{decision has no dependencies} \\ \frac{1}{1+r_i}\sum_{i=1}^{k} p_i \cdot Value(v_i) & \text{otherwise} \end{cases}$$

where the sum is over all of the edges (v, v_i) emanating from v and p_i is the probability of edge (v, v_i) and the corresponding decision being chosen. Given this representation, we use standard dynamic programing methods to compute the optimal decision with respect to a given decision tree. For the cloning mechanisms this implies a cloning timing which is optimal with respect to the available information regarding future loads.

4 The Cloning Algorithm

Among load balancing algorithms, one may find two main approaches [12]: overloaded processors that seek other, idle (or just random other) processors to let them perform part of the processes; idle (or lightly-loaded) processors that look for processes to increase their load. These approaches are sometimes combined with additional heuristics, or even merged. These two approaches can sometimes be utilized when designing a cloning algorithm for agents, however when cloning in open systems is discussed, considerable difficulties arise. Both approaches require that an agent locate other agents for task delegation. When using matchmaking agents, the first approach only requires that underloaded agents advertise their capabilities, and thus overloaded agents may contact them via matchmaking. Similarly, the latter approach requires that overloaded agents advertise their overload and required capabilities and resources. In addition, it requires that underloaded machines will be known to the overloaded agents as well, if there are no agents running on these machines (which means that the availability of their resources is not advertised). This information is not given in an open system. It could be provided if each machine the system is allowed to use runs an agent whose sole task would be supplying such information. This leads to an undesirable overhead of communication and computation. Therefore we utilize the first approach. That is, we provide algorithms appropriate for agents who perceive or estimate being overloaded.

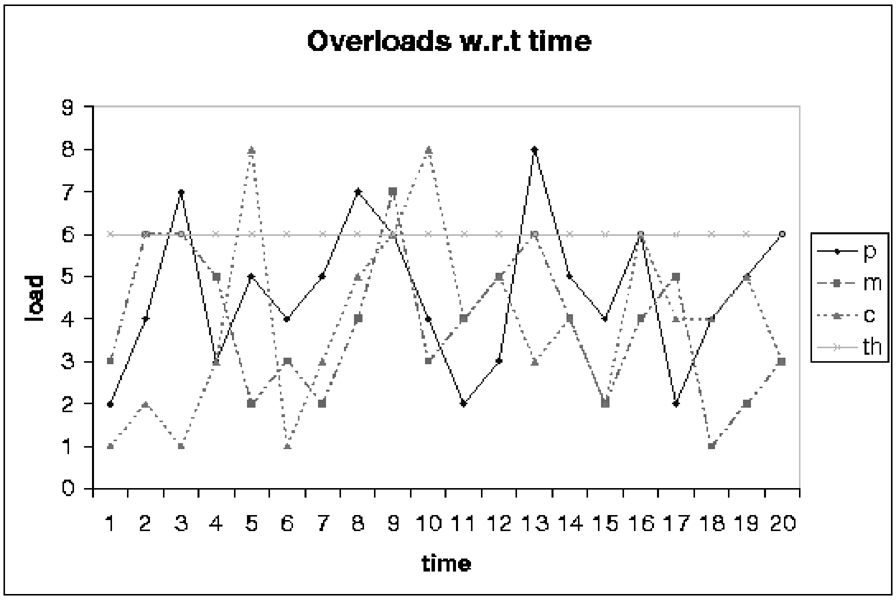

Figure 1: CPU, memory and communication loads

Below we provide an overview of the cloning procedure. In the appendix the reader may find some pseudo-code examples. The cloning procedure consists of the following components:

- Reasoning before cloning: includes the reasoning about the (possibly dynamic) task list with respect to time restrictions and capability and resource requirements. The consideration of the task list as well as agent capabilities, capacities, loads and machine loads results in a decision to clone or transfer tasks to already existing agents.

- Dividing the list of tasks: includes reasoning that considers the time intervals in which overloads are expected and accordingly selects tasks to be transfered. We illustrate the process of dividing the list of tasks by the following example. Suppose the current and future tasks have been scheduled. At each point in time, the required resources are the sum of the resources required for all of the tasks that were scheduled to be executed at this time. Figure 1 shows an example of the sums of three resources: cpu(p); memory(m); communication(m), with respect to time. The maximum capacity of the agent is depicted by the threshold horizontal line (th) leveled at 6. One can observe overloads whenever any type of demand for resources crosses this threshold. A periodic overload can be observed at times 3,8,13 with a period of 5 time units. Other overloads do not seem periodic. When attempting to prevent overloads, the agent first looks for tasks with a period that fits the period of the overloads and puts them in the list of candidate tasks for division. After re-computing the loads, it transfers one-shot tasks, if still necessary.

- Cloning: includes the creation and activation of the clone, the transfer of tasks, and the resulting inevitable updates of connections between agents via matchmaking. The following are the basic actions to be taken:

  – Create a copy of its code. This copy, however, may have to undergo some modification.
  – When cloning while performing a specific task, an agent should pass to its clone only the relevant sub-tasks and information which are necessary for the tasks passed to the clone. Otherwise, the clone may face the same overload problem as its creator. This is because the clone will have the same set of tasks as its creator had. Note that in contrast to the typical approach to agent migration [2], the cloning paradigm does not require the transfer of an agent state. The only transfer necessary is of the set of tasks to be performed by the clone.

- Reasoning after cloning: collects information regarding the benefits of the cloning and environmental properties (such as task stream distribution), and statistically analyzes them, as a means of learning for the future cloning.

While the reasoning of whether to initiate cloning is performed continually (i.e., when there are changes in the task schedule or if previous attempts to clone have failed), the cloning itself is a one-shot procedure.

5 Simulation

To examine the properties of the cloning mechanism and its advantages, a simulation was performed. The simulation results show that cloning increases (on average) the performance of the multi-agent system. In more detail, the performance enhancement as a result of cloning (if any) outweighs the efforts put into cloning.

The method of simulation was as follows. Each agent was represented by an agent-thread that simulated the resource-consumption and the task-queue of a real agent. The simulated agent has_a reasoning-for-cloning method which, according to the resource-consumption parameters and the task-queue, reasons about cloning. As a result of this reasoning, it may create a clone by activating another agent-thread (either locally or remotely). Information collected during the simulation is the usage of CPU and the memory and communication consumption of the agents. Each agent-thread receives a stream of tasks according to a given distribution. For each task it creates a task-object that consumes time and memory and requires communication. Some of these task-objects are passed to the clone agent-thread. The simulation was performed with and without cloning, to allow comparison. An agent-thread in the simulation must be subject to CPU, communication and memory consumption similar to those consumed by the agent it models in the MAS. Such information was collected from the real agent-system (RETSINA [6]) prior to the simulation. We have measured the resource consumption of the various types of RETSINA agents (see figure 2), when running 1 or 3 agents on each machine (when relevant, referred to by parentheses). The platforms on which these agents were examined are Sun Ultra-1s with 64MB, running Solaris. As one can observe, when running alone, all types of agents consume 40% to 45% of the CPU,

Agent type	Mem. size	CPU(1)	CPU(3)	Comm.(1)	Comm.(3)
Matchmaker	7.5 MB	43%	18-20%	94%	42%
Information	9.5-12 MB	45%	18-22%	94%	42%
Task	7 MB	44%	18-20%	94%	12%
Interface	9.6 MB	41%	18-20%	94%	15%

Figure 2: Resource consumption

whereas when running 3 agents on the same machine each consumes around 20% of the CPU. Not surprisingly, this results in a slower task performance. The same effect holds for the consumption of communication resources.

5.1 Simulation parameters and results

We simulated the agent system with and without cloning, with the following settings:

- Number of agents: 10 to 20.
- Number of clones allowed: 10.
- Number of tasks dynamically arriving at the system: up to 1000.
- Task distribution with respect to the required capabilities and resources for execution: normal distribution, where 10% of the tasks are beyond the ability of the agents to perform them within their particular deadlines.
- Agent capacity: an agent can perform 20 average task (i.e., requires average amount of resources) simultaneously.

The results of the simulation are depicted in figure 3. The graph shows that for small numbers of tasks (0 to 100) a system which practices cloning performs (almost) as well as a system with no cloning (although difficult to see in the graph, the performance is slightly lower due to the reasoning costs). However, when the number of tasks increases, the cloning system performs much better. Nonetheless, beyond some threshold, (around 350 tasks) even the cloning cannot help. Note that in the range 150 to 350 tasks cloning results in task performance which is close to the optimal (85% as compared to 90% which is optimal), where optimality refers to the case in which all of the available resources are efficiently used for task performance.

6 Related work

Methods for procedure cloning [3] and object cloning [10] were presented in the software engineering literature. The first serves as means for inter-procedural optimization, and the latter is used for eliminating parametric polymorphism and minimizing code duplication to overcome some typical inferior performance of object-oriented programs. Agent cloning is

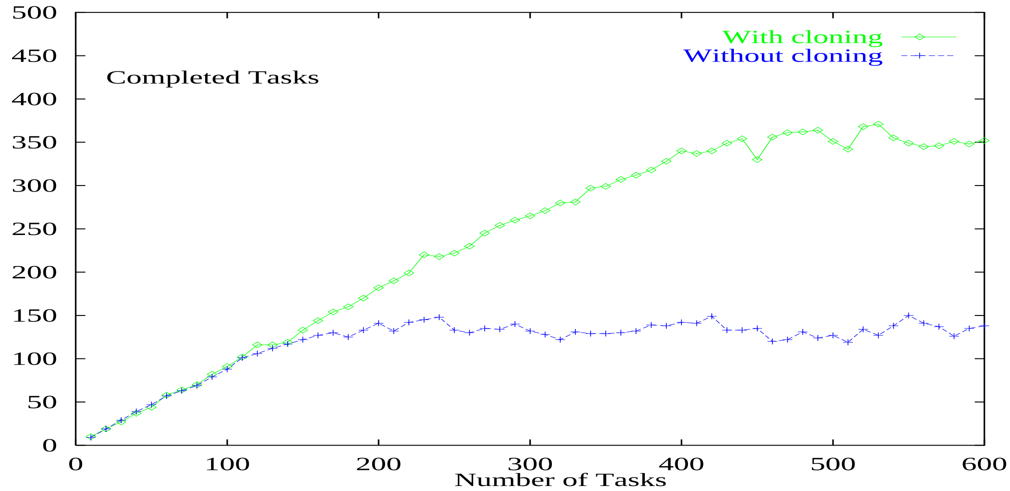

Figure 3: Task execution with and without cloning

performed differently and has a different aim. Although it attempts to improve the system's performance (as other cloning paradigms do), it concentrates on balancing the work loads of agents, not on the other computational issues. In addition, it allows for agent mobility.

The issue of agent load balancing was previously studied in [11], where the authors concentrate on multi-agent reinforcement learning in the context of adaptive load balancing. The MAS dealt with in that research was dynamic in the sense that resources and tasks were given probabilistically, and the agents had to efficiently allocate resources to tasks in order to optimize the global resource usage, while ensuring fairness. The underlying load balancing of the cloning paradigm has some similarities, however it is conceptually different. Cloning aims at optimizing the resource usage of the whole system. Our simulation results show that to some extent this was achieved. However we do not address the issue of fairness. More significantly, we deal with a system in which, in addition to resource and task dynamics, agents may dynamically appear and disappear, thus increasing the complexity of the load balancing. Finally, while the agents in [11] attempt to adapt their resource selection to the behavior of other agents (using reinforcement learning), our agents attempt to either delegate tasks to other existing agents or create other agents that will perform the overloading tasks.

In [9] different approaches to agent mobility are discussed, concentrating on messengers, which are mobile threads of execution who coordinate without central control. The salient features of messengers are their mobility, the creation of new messengers at run-time and collaboration and dynamic formation of sets of messengers for this collaboration. These properties seem quite similar to our requirements for MAS with cloning. However there is a major difference - messengers use a shared memory and rely on it for their functioning. This implies a strong restriction on their autonomy, which is unacceptable in MAS. Nevertheless, the π-calculi presented for messengers may be used to describe cloning MAS as well.

Mobile agents are an approach to remote computing which, in contrast to remote procedure calls, allows for sending the procedures to be performed on the remote host [14]. The procedure as well as its state are transported, and processing on the remote host is performed under pre-given authentication constraints. Cloning supports remote computing as well (however does not require it), but does not require the transmission of a procedure (or agent) state. This property significantly simplifies the performance of remote computing (especially due to the complexity encapsulated in state transmission).

7 Conclusion

Agent cloning is the action of creating and activating a clone agent (locally or remotely) to perform some or all of an agent's tasks. Cloning is performed when an agent perceives or predicts an overload, thus increasing the ability of a MAS to perform tasks. We have presented agent cloning as a means for balancing the loads and improving the task performance of a MAS running on several remote machines. We provided explicit methods of implementation and tested these methods by simulation. We found that for large numbers of tasks, cloning significantly increases the portion of tasks performed by a MAS. In a MAS where tasks require information gathering on the web (e.g., RETSINA), the additional reasoning needed for cloning is small compared to task execution requirements. Currently we are in the process of embedding the cloning protocol into each autonomous agent in the RETSINA MAS. In addition, we are developing protocols for agent merging or self-extinction.

References

[1] D. Chess, B. Grosof, and C. Harrison. Itinerant agents for mobile computing. Technical Report RC 20010, IBM Research Division, 1995.

[2] R. Clark, C. Grossner, and T. Radhakrishnan. CONSENSUS and COMPROMISE: Planning in cooperating expert systems. *Int. Journal of Intelligent and Cooperative Information Systems (submitted).*

[3] K. Cooper, M. Hall, and K. Kennedy. Procedure cloning. In *Proceedings of the International Conference on Computer Languages*, pages 96–105. IEEE Computer Society, April 1992.

[4] K. Decker, K. Sycara, and M. Williamson. Inteligent adaptive information agents. In *Proceeding of AAAI-96 workshop on Intellignet Adaptive Agents*, Portland, Oregon, 1996.

[5] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceeding of IJCAI-97*, Nagoya, Japan, 1997.

[6] Keith Decker, Anandeep Pannu, Katia Sycara, and Mike Williamson. Designing behaviors for information agents. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents*, New York, 1997. ACM Press.

[7] Robert S. Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. Mobile agents for mobile computing. Technical Report PCS-TR96-285, Dartmouth College, Computer Science, Hanover, NH, May 1996.

[8] M. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. In *Proceedings of ACM Computer Network Performance Symposium*, April 1982.

[9] G. Di Marzo, M. Muhugusa, and C. Tschudin. Survey of theories for mobile agents. Working paper, The Computing Science Center, University of Geneva, Switzerland, November 1995.

[10] J. Plevyak and A. Chien. Type directed cloning for object-oriented programs. In *Proceedings of the workshop for Languages and Compilers for Parallel Computers*, Columbus, Ohio, 1995.

[11] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: a study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

[12] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, editors. *Sceduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press, New York, 1995.

[13] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, pages 36–45, December 1996.

[14] J. E. White. Telescript technology: Mobile agents. General Magic White Paper, 1996.

8 Appendix: Cloning Pseudo-Code Examples

Parts of the protocol are given below in pseudo-code. Arrays are denoted by `[]` and indexed by time, and procedures are denoted by `()`.

```
// Main reasoning and cloning protocol:
for each time interval t{
 overloadAt[t] = reasonAboutOverloads(t); //current and future
 if (overloadAt[t]) canPassTasks[t] = findUnderloadedAgents(t);
 if (canPassTasks[t]){
  reasonForTaskSplit(); transferTasks();}
 else if (canCloneLocally()){
  cloneLocally(); reasonForTaskSplit(); transferTasks();}
 else if (canCloneRemotely()){
  cloneRemotely(); reasonForTaskSplit(); transferTasks();}
 else sorry(''can't split or clone'');}
reasonAfterCloning();

// The reasoning methods:

reasonAboutOverloads(t){
getCurrentTaskList(); getExpectedTaskFlow(t);
requiredAt[t] = calculateRequiredResourcesAt(t);//current and future
selfCapacity = getSelfCapacity(t);
if (selfCapacity >= requiredAt[t]) return FALSE; else return TRUE;}

findUnderloadedAgents(){
locateAgentsWithAppropriateCapabilities();//contact a matchmaker
for each agent found{
 agentLoads = acquireAgentLoads()//query the agent for this info.
 If (agentLoads at t > requiredAt[t] - selfCapacity - resourcesForCloning)
  add agent to underloadedList;}
return underloadedList;}

reasonForTaskSplit(){
for each t where (overloadAt[t] = TRUE)
 for each task in taskList,   if (timeInterval(task) = t), add task to prospectiveTasks;
periods = periods of periodic tasks in prospectiveTasks;
if (periods ≠ ∅), for each period p in periods{
 if (overloadAt[·] has a period p) move one corresponding task
  at a time from taskList to taskSplit;
 for each t, recompute overloadAt[t];}//some will become FALSE
else
 for each t where overloadAt[t] = TRUE{ //until none are true
  move one task (at a time) with timeInterval = t to taskSplit;
  recompute overloadAt[t];}}
```

Figure 4: Cloning Pseudo Code Examples