Adaptive Task and Resource Allocation in Multi-Agent Systems

S. Shaheen Fatima and Michael Wooldridge
Department of Computer Science
University of Liverpool
Liverpool L69 7ZF, U.K.
{s.s.fatima, m.j.wooldridge}@csc.liv.ac.uk

ABSTRACT

In this paper, we present an adaptive organizational policy for multi-agent systems called TRACE. TRACE allows a collection of multi-agent organizations to dynamically allocate tasks and resources between themselves in order to efficiently process an incoming stream of task requests. TRACE is intended to cope with environments in which tasks have time constraints, and environments that are subject to load variations. TRACE is made up of two key elements: the task allocation protocol (TAP) and the resource allocation protocol (RAP). The TAP allows agents to cooperatively allocate their tasks to other agents with the capability and opportunity to successfully carry them out. As requests arrive arbitrarily, at any instant, some organizations could have surplus resources while others could become overloaded. In order to minimize the number of lost requests caused by an overload, the allocation of resources to organizations is changed dynamically by the resource allocation protocol (RAP), which uses ideas from computational market systems to allocate resources (in the form of problem solving agents) to organizations. We begin by formally defining the task allocation problem, and show that it is NP-complete, and hence that centralized solutions to the problem are unlikely to be feasible. We then introduce the task and resource allocation protocols, focussing on the way in which resources are allocated by the RAP. We then present some experimental results, which show that TRACE exhibits high performance despite unanticipated changes in the environment.

1. INTRODUCTION

It has long been recognized that the ability of a multi-agent system to dynamically reorganize its structure and operation at run-time is highly valuable for many application domains; see, e.g., [2, 11, 10]. In this paper, we investigate the performance of a particular approach to dynamically reallocating tasks and resources in a multi-agent system. This adaptive organizational policy ([1]) is called TRACE (*T*ask and *R*esource *A*llocation in a *C*omputational *E*conomy). TRACE was designed for systems with the following characteristics:

- tasks — problems to be solved — arrive in the system at unpredictable, random intervals, and so the system load varies over time;
- tasks are characterized by a type (corresponding to the skills required by an agent in order to carry the task out), a duration (the amount of time it takes to carry the task out), a deadline (the latest time at which the task can be completed), and a priority (how critical the task is); and
- agents are heterogeneous, in the sense that different agents have different capabilities.

The main aim is thus for agents in the system to be able to schedule tasks on the fly so as to maximize the number of tasks successfully completed by their deadline. As agents have different capabilities, it is necessary for agents to cooperate in order to allocate tasks effectively.

The remainder of this paper is structured as follows. We begin in the following section by describing and formally defining the problem that TRACE is intended to solve. We then investigate its computational complexity, and show that the underlying problem — multi-agent task scheduling — is NP-complete, and hence is unlikely to be amenable to centralized, brute force solutions. In section 3, we present TRACE in detail. The TRACE algorithm builds upon the previous work of Fatima et al [9, 8]: the problem of developing an adaptive organizational policy is divided into task and resource allocation sub-problems. In TRACE, changes in load are handled by changing the distribution of knowledge across agents and diverting resources where they are needed most. This entails *dynamic reorganization* of the system: the structure of the system changes over time, with agents entering and leaving problem solving organizations. Existing approaches such as agent cloning [4] and the mobile agent paradigm [13] do not allocate resources to tasks on priority basis. In order to do this, our approach uses a price-directed micro-economic approach [17, 6, 16, 15] for resource allocation. TRACE has two components: the task allocation protocol (TAP) and the resource allocation protocol (RAP). The TAP takes requests for tasks and the schedules that agents have, and cooperatively allocates subtasks to agents within an organization. As requests arrive arbitrarily, at any instant, some organizations could have surplus resources while others could become overloaded. In order to minimize the number of lost requests caused by an overload, the allocation of resources to organizations is changed dynamically by the resource allocation protocol (RAP), which uses ideas from computational market systems to allocate resources (in the form of problem solving agents) to organizations.

The MAS organization for TRACE is described in section 2. The task and resource allocation protocols are described in section 3. The results of simulation are presented in section 4, and section 5 gives some conclusions.

2. THE PROBLEM

In this section, we formally define the problem that the TRACE system is trying to solve, and the structure of the system. The basic idea is as follows. We have a multi-agent system $\mathcal{M}$, which is structured as a set of k computational organizations $\mathcal{M} = \{o_1, \ldots, o_k\}$. Each organization is a set of agents: organizations have no internal structure. Different organizations contain mutually disjoint sets of agents. We denote the set of all agents by $\mathcal{AG} = \{a_1, \ldots, a_l\}$. The members of an organization $o \in \mathcal{M}$ are denoted by $\mathit{membs}(o)$, so $\mathit{membs}(o) \subseteq \mathcal{AG}$. If $\mathcal{M}$ is a multi-agent system, then we will abuse notation slightly and write $\mathit{membs}(\mathcal{M})$ to denote the members of $\mathcal{M}$, so $\mathit{membs}(\mathcal{M}) = \bigcup_{o \in \mathcal{M}} \mathit{membs}(o)$.

At every time step, agents may be allocated a (possibly empty) set of *tasks* to carry out. (We are not concerned here with where these tasks arise from.) A task τ is defined as a quad

$$\tau = \langle t, dl, dur, prior \rangle$$

where t is the *type* of the task, $dl \in I\!\!N$ is a natural number corresponding to the *deadline* of the task, $dur \in I\!\!N$ is a (non-zero) natural number denoting the *duration* of the task, and finally, $prior \in I\!\!N$ is a natural number denoting the *priority* of the task.

The type of a task corresponds to the type of activity that the task involves. The idea is that different tasks may require different kinds of capabilities in order to carry the task out; different agents have different capabilities, and so not all agents can carry out all types of tasks. The type of a task is thus a specification of the capabilities required to carry out the task.

The deadline of the task corresponds to the latest time at which the task might usefully be carried out. We model time via the natural numbers $I\!\!N$, and assume that every agent uses the same model, and in addition, that every agent has access to a global system clock. Notice that an alternative approach would be to specify a quality function for each task, which indicates the utility of completing the task at each time point (cf. [3]).

The duration of a task indicates the number of time steps required to carry out the task. In order to carry out a task, an agent with the appropriate capability must work on the task for at least this many consecutive time steps in order to complete it.

Finally, the priority of a task simply denotes how important the task is; the higher the priority, the more important it is.

Let $\mathcal{TY} = \{t_1, \ldots\}$ be the set of all task types, and let

$$\mathcal{TA} = \mathcal{TY} \times I\!\!N \times I\!\!N \times I\!\!N$$

denote the set of all tasks. If $\tau \in \mathcal{TA}$ is a task, then we denote the type of τ by $\mathit{type}(\tau)$, the deadline of τ by $\mathit{dl}(\tau)$, the duration of τ by $\mathit{dur}(\tau)$, and the priority of τ by $\mathit{prior}(\tau)$.

Agents each have a set of *capabilities* and a *schedule*. An agent's capabilities are a subset of $\mathcal{TY}$; we denote the capabilities of agent $a \in \mathcal{AG}$ by $\mathit{cap}(a)$, so $\mathit{cap}(a) \subseteq \mathcal{TY}$. If $a \in \mathcal{AG}$ and $\tau \in \mathcal{TA}$, then agent a has the capability to carry out task τ iff $\mathit{type}(\tau) \in \mathit{cap}(a)$.

An agent's schedule defines what tasks the agent is working on and when it is working on these tasks. Tasks on an agent's schedule represent *commitments* that the agent has about the future. Formally, a schedule is a function from time points to tasks; we denote the schedule of agent $a \in \mathcal{AG}$ by $\mathit{sched}(a)$, so $\mathit{sched}(a)$ is a (partial) function $\mathit{sched}(a) : I\!\!N \to \mathcal{TA}$. Notice that schedules must satisfy the following requirement:

$$\forall a \in \mathcal{AG}, \forall n \in I\!\!N, \mathit{sched}(a)(n) = \tau \text{ implies } \mathit{type}(\tau) \in \mathit{cap}(a).$$

Thus agents are only allowed to be committed to tasks of which they are capable.

A schedule is *complete* if every task that appears in the schedule is allocated sufficient consecutive time steps to completely carry out the task, where the number of time steps required to complete the task is defined by the duration attribute of the task. Notice that an agent cannot process more than one task at any given time.

Within each organization, member agents are divided into two classes: *permanent agents* and *marketable agents*. Permanent agents are agents that are permanently assigned to a particular organization. In contrast, marketable agents are agents that may be temporarily re-assigned to another organization. Intuitively, permanent agents represent the *capital* of the organization, whereas marketable agents represent the *labour*.

If $o \in \mathcal{M}$ is an organization, then we denote the permanent agents in o by $\mathit{perm}(o)$, and the marketable agents in o by $\mathit{mar}(o)$; we require that $\mathit{perm}(o) \cap \mathit{mar}(o) = \emptyset$ and that $\mathit{perm}(o) \cup \mathit{mar}(o) = \mathit{membs}(o)$.

Permanent agents in each organization periodically receive tasks to carry out — we refer to these as requests. These requests arrive randomly; no agent knows either what tasks it will receive or when it will receive tasks. The intuition is that these tasks are problems that the agent must solve. If an agent a cannot solve a task τ itself (i.e., if either $\mathit{type}(\tau) \notin \mathit{cap}(a)$ or else the schedule of a does not permit completion of τ before $\mathit{dl}(\tau)$), then a can ask other agents in its organization to perform the task. These agents are assumed to be benevolent, so that if they have the capability and the opportunity, then they will agree to carry out the task. However, if no other agent in the organization can complete the task before $\mathit{dl}(\tau)$, then the task is *decommitted*, in which case it reappears in the system at a later date, with a later deadline. One of the main goals of our work is to minimize the number of decommitted tasks.

The system is assumed to proceed in a series of rounds, where at every round, an agent may receive tasks to process, and needs to decide how to process them. The agent may communicate with other agents at this stage, if required. After deliberation, the agents are required to update their internal schedule, and in particular, to come to a decision about what task to work on for that round (they cannot subsequently backtrack from the decision about what to work on for that round, although they are allowed to modify their later schedule).

If an organization finds that it has tasks than it cannot process, then it can *buy* labour from other organizations in order to subsequently process tasks. This labour comes in the form of the marketable agents — permanent agents are not permitted to work for other organizations. Labour is purchased for a single round. In order to purchase labour, each permanent agent a is given a *fund* $F_a \in \mathbb{R}$, which is renewed at every round.

Complexity

It is worth noting that the task scheduling problem TRACE is attempting to solve is computationally hard; in fact, we can show that the problem of determining whether or not it is possible to allocate a collection of tasks to a group of agents such that all tasks are guaranteed to be completed is NP-complete. Consider a multi-agent system as specified above, and consider an allocation of tasks to agents; recall that every agent will receive a (possibly empty) allocation of tasks at every round. Formally, such an allocation can be understood as a function with the signature

$$\mathcal{AG} \to 2^{\mathcal{TA}}$$

We will assume that the function is presented in the form of a table. The idea is to determine whether or not, given the agents with capabilities as specified and current schedules, all tasks can be successfully completed before their deadline. Assuming that agents have finite schedules (i.e., that there is some time in the future after which they have no tasks scheduled) then we can show:

THEOREM 1. *Multi-agent task scheduling is* NP*-complete.*

PROOF. *For membership of* NP*, simply guess a schedule for each agent, and verify that this schedule is complete (i.e., every task is fully carried out), that every task is carried out by an agent with the appropriate capabilities, and that every task is finished before its deadline. The size of the schedule to be guessed will be bounded by the deadline of the last scheduled task, the number of agents, and the number of tasks. Verification can easily be done in time polynomial in the size of the schedule.*

To show NP*-hardness, we reduce the* bin packing problem *to multi-agent task scheduling [12, pp204–206]. An instance of the bin packing problem is determined by n positive integers, $v_1, \ldots, v_n$, and two further integers, the capacity C and the number of bins B. The goal is to determine whether the numbers can be partitioned into B subsets, each of which has total sum at most C. To reduce bin packing to multi-agent task scheduling, we create a task τ_i corresponding to each integer v_i, with deadline $dl(\tau_i)$ set equal to C and duration $dur(\tau_i)$ set equal to v_i. The type of each task is set to some dummy value t_0. We then then create B agents, (i.e., one for each bin), where each agent is capable of tasks of type t_0 (so any task can be given to any agent). It should be clear that the tasks can be partitioned between the agents so as to be completed before their deadlines just in case the integers $v_1, \ldots, v_n$ can be partitioned into B bins so that each bin has total value at most C. Since the reduction is clearly possible in polynomial time, we are done.* $\square$

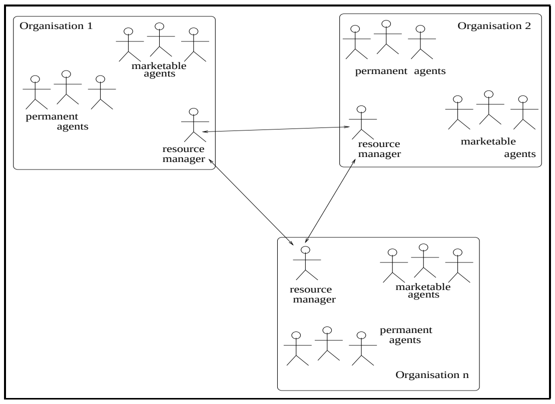

Figure 1: Structure of a multi-agent system in TRACE.

Notice that this problem is somewhat similar to the COOPSAT problem discussed in [5], where the aim is for an agent to determine whether or not cooperation is in principle possible in order to solve one of its goals.

Also note that the version of this problem which involves determining a general policy for task allocation that can be used at run time to allocate tasks between agents is more akin to stochastic scheduling and other PSPACE-complete problems of decision making under uncertainty — see [12, pp470–474].

3. THE TRACE SYSTEM

We can now describe the TRACE system in more detail. The aim of the system is to maximize the throughput of tasks, and in particular:

- to process tasks before their deadline;
- to minimize the number of decommitted tasks;
- to process tasks on a priority basis; and
- to make efficient use of resources.

The overall structure of the TRACE system is shown in Figure 1. A MAS in TRACE is a collection of computational organizations, where each organization has three components:

- a set of *permanent agents*, which are assigned to the organization indefinitely;
- a set of *marketable agents*, which are "owned" by the organization, but which may be temporarily hired by other organizations in order to make up shortfalls in labour;
- a distinguished agent known as the *resource manager* (RM), which has the dual responsibility of both determining resource needs (in the sense of agents) for the organization, and of renting labour from other organizations when required.

Task requests are sent to members of the organization arbitrarily, and so the requirement for resources at each organization varies, leaving some organizations with surplus resources, and others with insufficient resources. In order to schedule tasks, TRACE makes use of two protocols — the task allocation protocol (TAP), and the resource allocation protocol (RAP). Allocation of tasks to agents within an organization is done through the TAP and allocation of resources to each of these organizations is done through the RAP. We now describe each of these protocols in turn.

3.1 Task Allocation Protocol

The task allocation protocol used in TRACE is closely based on the Contract Net [14]. It is used within an organization in order to determine an allocation of tasks to agents within the organization. This involves finding suitable team members and the actual time at which the tasks can be executed.

Due to lack of space only a high level description of the TAP is presented here — see [7] for details. In order to find team members and arrive at a commonly agreed time with them, an agent a that receives a request for a task τ firstly determines whether or not it has the capability to carry out the task (i.e., whether or not $type(\tau) \in cap(a)$), and if so, whether the task can be carried out by the agent before the deadline[1] $dl(\tau)$ of the task. If not, it generates a proposal for others to carry out the task, and sends an announcement message [14] to all agents of its organization indicating the task and the proposed time of the task. The other agents of the organization determine if the task and its time are acceptable with reference to their existing schedule, and if so send a bid message to the agent that sent the announcement — the *organizer*. If the proposed time is not acceptable, the prospective team member sends a bid with a modified time. If the organizer finds the modified time acceptable, then it agrees in principle to the task, otherwise it may propose some other time and this process repeats until a mutually agreed time is arrived at.

During the task allocation process, any conflicts that arise with preexisting commitments are resolved on the basis of task priorities. Lower priority tasks are either rescheduled to accommodate a more critical task, or decommitted altogether if deadlines make rescheduling impossible. Whenever an agent drops an existing commitment to accommodate a higher priority request, it informs all organization members. When a task is decommited, the manager of that task increments a counter, which indicates the number of decommitted tasks. All task managers periodically send this value to their respective RMs. In addition to this, all marketable agents calculate the percentage time they remain idle during a certain period. This information is also passed on to the respective RMs, which then perform reallocation.

(As an aside, notice that in our actual implementation of TRACE, agents were sent *goals* to achieve. In order to satisfy these goals, agents made use of a predetermined plan library, in which each plan was a sequence of tasks to carry out. Instead of delegating a single task, a task organizer would negotiate with members of its organization in order to delegate all sub-tasks. See [7] for details.)

3.2 Resource Allocation Protocol

As the demand for tasks of different types varies over time, so the need for labour resources will vary in different organizations. The RAP periodically reallocates marketable agents to organizations in accordance with their demands. This results in reorganization (change in the number of agents in an organization, the distribution of domain knowledge and the communication structure) of the MAS. The RAP reorganizes the MAS so that decommitted requests can be honored when they arrive again.

The RM obtains the resource needs of an organization from its permanent agents, and on the basis of this information, arrives at a suitable allocation. As the number of marketable agents is fixed, and multiple organizations could be contending for these agents, an allocation is arrived at on the basis of the criticality of decommitted tasks. The permanent agents of an organization convey information about the criticality of decommitted tasks indirectly by contributing some funds to the resource manager; the greater the criticality of decommitments, the higher the contribution of funds. The permanent agents also specify how many additional agents (μ_o) would be required by the organization $o \in \mathcal{M}$. The method used for obtaining μ_o is explained below. Thus the contribution of funds made by an organization indicates the maximum price that the organization is willing to pay in order to buy μ_o marketable agents. The contribution of funds varies from organization to organization and reflects their relative needs for additional resources. The organizations that offer more funds are considered to be more in need of resources than those offering less.

The RMs periodically determine the resource needs of their respective organizations and conduct reorganization accordingly. Each such period is called *reorganization cycle*. The MAS is organized as a market economy composed of interacting buyers and sellers. The commodities in this economy are the marketable agents required to carry out tasks. Buyers are organizations that wish to purchase new agents in order to perform some computation. Sellers are the resource managers that wish to sell the marketable agents for the duration of one reorganization cycle. The buyers and sellers execute a resource allocation protocol to arrive at an optimal allocation of resources. Reallocation is done at the beginning of every reorganization cycle. For reallocation to be completed, each RM must perform the following steps:

1. Obtain information pertaining to the requirement for additional resources from agents of its organization.

2. Compute the *equilibrium allocation* on the basis of information acquired in step 1.

3. Transfer relevant domain knowledge to the newly allocated agents.

4. Notify permanent agents of its organization about the new allocation.

Each of these steps is explained in detail below.

Step 1. The requirement for resources in any reorganization cycle is determined on the basis of the information about the previous cycle. Agents of an organization convey the following four items of information about the previous cycle to their respective RMs at the beginning of every reorganization cycle:

1. *Information about the number of decommitments (D).*

[1]Deadlines are associated with tasks to ensure termination of the TAP. Thus if a task request is made without a deadline, TRACE fixes a deadline for it on the basis of its duration. These are low priority tasks that are assumed to be resubmitted if they get decommitted.

The permanent agents send this information because only they act as organizers and keep track of the number of decommitments. Demand for new agents, μ_o, in an organization o can be computed from the number of decommitments (D) made by the organization in that cycle and the number of tasks that remain uncompleted. As the requests that are decommitted by the TAP are the ones that have low priority, they are very much likely to occur again. Based on this information, new agents are introduced that have the capability to take on the decommitted tasks when they are requested again.

2. *Information about the decommitted tasks.*

   The type, duration, and deadline of decommitted tasks is also sent by each permanent agent to the RM; this information is used for dynamic distribution of domain knowledge to agents (see step 3).

3. *Information about idle time.*

   This is used to identify idle agents. Marketable agents convey this information, because only these agents can be reallocated. The ones that remain idle for more than 50% of the reorganization cycle time are treated as superfluous and considered for allocation to some other organization in need of them. Thus if an organization o is allocated X marketable agents in a cycle, has $D = 0$ for that cycle, and has Y agents that remain idle most of the time, then the number of agents it requires for the next cycle, μ_o, is taken to be $X - Y$.

4. *Information about the contribution of funds (F_a).*

   Each permanent agent a in organization o contributes funds F_a to their resource manager in every reorganization cycle. The sum F_o of these values for all permanent agents indicates the maximum the organization is willing to pay for buying μ_o additional agents (obtained from item 1 or 3).

Funding units are used as an abstract form of priority. The organizations that contribute more are deemed to be more in need of additional agents than the ones contributing less. It is assumed that the amount of funds to be contributed is determined by the application. It is the application's burden to ensure that important computations are well funded. The RMs conduct markets (see step 2) on behalf of high-level applications and intimate the equilibrium price of a marketable agent to the permanent agents of its organization. On the basis of its funding in any reorganization cycle, the equilibrium price for that cycle, and the total number of decommitments, the application can determine how much to contribute for the next cycle.

Step 2: Every RM encapsulates details about the funds (from item 4) and demand for new agents (from items 1 or 3) for its organization and communicates this information to every other RM of the MAS. The protocol consists of this communication step followed by a local computation by each resource manager. Each resource manager locally computes the equilibrium price of a marketable agent.

The demand for marketable agents at organization o, at price p, is given by the function $z_o(p)$, which is defined as

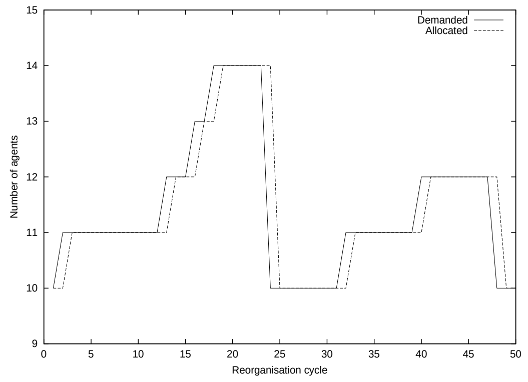

Figure 2: A graph of agents demanded (solid line) and agents allocated (dotted line) against time during a typical run of TRACE.

$$z_o(p) = \begin{cases} \mu_o & \text{if } p \leq \frac{F_o}{\mu_o} \\ \frac{F_o}{p} & \text{otherwise} \end{cases} \qquad (1)$$

where $F_o = \sum_{a \in perm(o)} F_a$ is the contribution of funds made by the organization o.

The total demand for agents at price p across all the organizations in system $\mathcal{M}$ is:

$$\sum_{o \in \mathcal{M}} z_o(p)$$

Let $s_{\mathcal{M}}(p)$ denote the total supply of marketable agents at price p in $\mathcal{M}$:

$$s_{\mathcal{M}}(p) = \begin{cases} |\bigcup_{o \in \mathcal{M}} mar(o)| & \text{if } p \geq p_{min} \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

where p_{min} indicates the minimum price at which the resource managers will sell marketable agents.

The market will be in equilibrium when p has a value such that

$$\sum_{o \in \mathcal{M}} z_o(p) = s_{\mathcal{M}}(p) \qquad (3)$$

In order to find the value of p that satisfies (3), the RMs initialize p to the maximum of prices offered by all the organizations. This price is then iteratively decreased till Equation (3) is satisfied. The number of iterations can be reduced by using an approximation condition of the form

$$\left| \sum_{o \in \mathcal{M}} z_o(p) - s_{\mathcal{M}}(p) \right| \leq \epsilon \qquad (4)$$

where

$$\left| \sum_{o \in \mathcal{M}} z_o(p) - s_{\mathcal{M}}(p) \right|$$

is the aggregate excess demand. A more precise statement of the computation is now given.

	Number of Organizations				
	4	8	16	32	64
Percentage reduction	74.12	75.98	74.30	75.10	74.10

Table 1: TRACE scales up: the percentage reduction in decommitments remains stable as the number of organizations grows.

	Org_1	Org_2	Org_3	Org_4
Funding Ratio	33.23	23.33	20.00	23.22
Proportion of agents allocated	34.21	23.68	18.42	23.68

Table 2: Fairness of resource allocation.

Each resource manager goes through the following computations:

1. Communicate F_o and μ_o for its organization to every other resource manager.

2. Initialize p to the maximum of prices offered by all the organizations, and compute the aggregate excess demand.

3. Iteratively decrement price p by δ_p (the *step size parameter*) until the approximation condition (4) is satisfied.

Once the equilibrium price is determined the RMs transfer relevant domain knowledge to the newly allocated agents. This is done as follows.

Step 3. The steps described above compute the equilibrium allocation of marketable agents to organizations. However, these agents can lack the domain knowledge required to take on the tasks of an organization. This means that such agents need to be first endowed with the required knowledge before they are allocated to an organization. In addition to managing resource allocation, the RM also does the job of allocating this knowledge to the new agents.

The RM possesses all the knowledge required by its organization. Out of this only a selected portion is allocated to the new agent. In order to determine this portion the RM obtains information about the decommitted tasks from all agents of its organization (see step 1). The domain knowledge required to execute these tasks is then transferred to the incoming agent. This kind of dynamic distribution of knowledge enables effective use of available computational resources; an agent that is idle but lacks knowledge required to execute tasks can acquire that information as indicated above.

Step 4. The RMs finally notify all permanent agents of their respective organizations about the equilibrium price and the new allocation. After reorganization, all permanent agents update their organizational knowledge. Subsequently, the TAP refers to this changed organizational knowledge to select team members for tasks. Organizations in TRACE therefore exist at a logical level and are represented as organizational knowledge with permanent agents.

4. EXPERIMENTAL RESULTS

In order to evaluate the effectiveness of TRACE, a number of experiments were carried out. These serve to quantify its ability to reduce the number of decommitments, fairly distribute resources among competing goals, adapt to changes in computational load by reorganizing the MAS, and make effective use of resources. The system was implemented in C, and the behavior of the system was studied by randomly varying the computational load across different organizations in the MAS.

Reduction in Decommitments. The first experiment was done to measure the reduction in the number of decommitments made by the system. Each organization of the MAS was assumed to have 10 permanent agents and the number of marketable agents was 10 times the number of organizations. The system was allowed to run for 100 reorganization cycles by gradually varying the computational load in each organization for every cycle — Figure 3 shows a fragment of a typical run. Different organizations contributed different amounts of funds but the amount contributed by an organization was held constant over all the 100 cycles. The total number of decommitments over the entire run was then found.

As a control, the experiment was repeated without using the TRACE RAP (i.e., by simply dividing the marketable agents equally among the organizations, and keeping the number of agents in each organization constant throughout). From these two results the percentage reduction in the number of decommitments using the reorganization method was determined. The results of this study are summarized in Table 1.

A desirable characteristic of open MAS is the ability to scale well to large systems. The simulation results as shown in Table 1 were obtained by increasing the number of organizations from 4 to 64. Note that in this experiment the number of agents increases with the number of organizations (we had 10 permanent agents in each organization, and the total number of marketable agents was 10 times the number of organizations). The number of requests was increased in the same proportion. Basically, the conditions to which an organization is subjected are the same but the number of such organizations has been scaled up.

This increase, however, did not affect the percentage reduction in decommitments. This indicates that with respect to reduction in decommitments, the proposed reorganization approach scales well to large systems. These results were obtained in the absence of a funding strategy. The performance of the system can however be improved by having the application make use of effective funding strategies.

Fairness of Resource Allocation. In TRACE, funds are

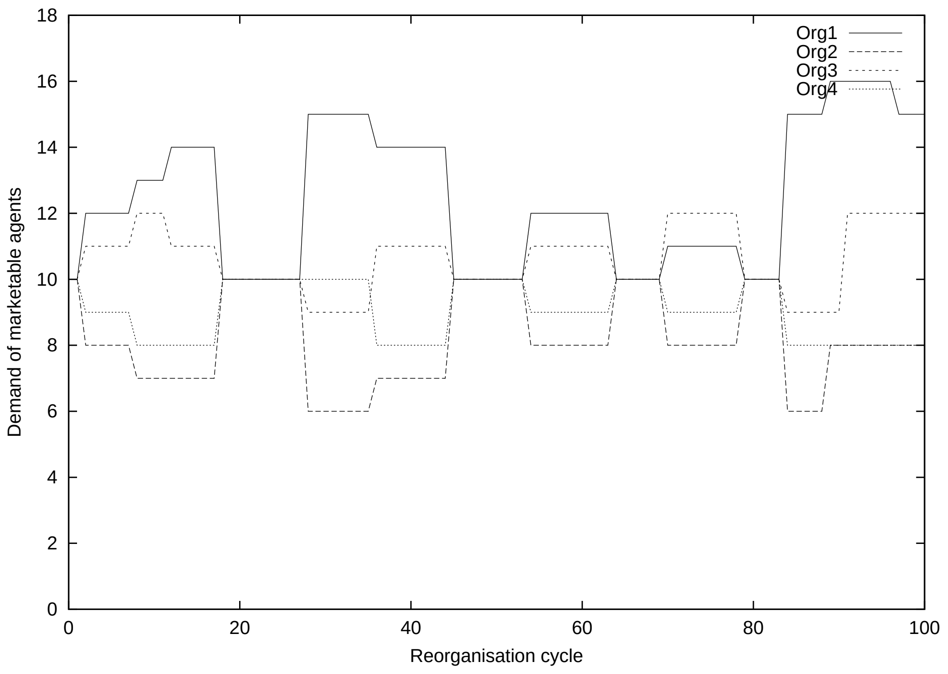

Figure 3: A graph of load against time for four organizations in a typical run of TRACE.

analogous to priority. In order to test the fairness of resource distribution, a set of experiments was done for different funding ratios among organizations (keeping the demand for agents constant across all of the organizations).

The results of these experiments are summarized in Table 2. The first row specifies the funding ratio and the second row indicates the relative number of agents obtained by each organization. A fair distribution is one in which each organization is able to obtain a share of resources that is close to its share of total system funding. As we can see from Table 2, TRACE allocates resources in a manner that is reasonably close to the funding ratio in all the runs.

Adaptiveness of the MAS. The main objective of TRACE is to make the MAS adaptive to variations in load. Figure 2 shows the variation in number of marketable agents demanded over 50 cycles in one organization and the agents acquired by it over 50 cycles through the RAP. As the number of decommitments in a cycle increases, the number of new agents in the next cycle increases correspondingly. For instance, in reorganization cycle 2, when the requirement for agents increased from 10 to 11, the number of additional agents in the next cycle increases correspondingly. Similarly, a decrease in the number of decommitments results in a decrease in the number of agents. These results demonstrate the ability of the MAS to adapt to load variations.

Efficient use of Resources. As shown in Figure 1, the average number of agents over 50 cycles is 12. In order to achieve the same level of performance as in TRACE, a MAS with constant number of agents would require 14 permanent agents (the maximum number of agents required by one organization). Thus the proposed approach, (which requires 12 agents on an average), is more economical in terms of resources.

Reorganization Overhead. Clearly, reorganization is not desirable if the computation and communication overheads outweigh the benefits accrued. One source of overhead is the resource managers, which were introduced into the multi-agent system to perform reallocation of agents. The sole function of these agents is to perform reallocation at the beginning of every reorganization cycle. The resource managers however remain idle for the remaining part of the cycle. In order to make effective use of these resource managers, they can be allocated tasks just like other agents of the organization. The second source of overhead is the communication cost that is incurred as a result of transfer of information from the agents of an organization to their resource manager and vice versa. The number of messages that are sent to a resource manager in organization o is equal to the number of agents in its organization, i.e., $|membs(o)|$. Thus there would be $O(|membs(o)|)$ transfers of information to the resource manager at the beginning of the reorganization cycle. After arriving at the equilibrium price, the resource manager broadcasts information about the new agents and the equilibrium price to all agents of its organization. This requires another $O(|membs(o)|)$ message transfers. So for every organization there would be $O(2 \times |membs(o)|)$ messages sent per reorganization cycle.

In addition to this, some communication takes place among resource managers. This is the communication regarding the funds and the required number of agents that is broadcast by every resource manager to every other resource manager. However, this communication is not considerable since the number of resource managers is very small compared to the total number of agents in the multi-agent system. The third factor that needs to be considered is the time required by resource managers to arrive at the equilibrium price. The convergence time depends on the number of iterations required to reach equilibrium. The number of iterations depends on δ_p (the step size parameter), and ϵ. The smaller these values, the more feasible is the allocation of agents, but larger is the number of iterations. Larger values reduce the number of iterations but may not result in an allocation with the same degree of feasibility: the price of an agent that resource managers arrive at may not be sufficiently close to the actual equilibrium price. This results in an infeasible allocation of resources where the amount of resources allocated may not be equal to the amount of resources available.

5. CONCLUSIONS AND FUTURE WORK

This paper proposed an adaptive organizational policy called TRACE for time constrained domains with varying computational load. The simulation results presented here show the effectiveness of TRACE.

A number of other approaches have been proposed for handling load variations in multi-agent systems. One of these is the work on organization self design [2, 11, 10]. This approach performs reorganization through two primitives, namely composition and decomposition of agents. Decomposition divides one agent into two and is triggered in response to an overload. Composition combines two agents into one in order to free up computing resources when they are not required. The goal of this method is to achieve real time performance through reorganization.

Mobile agents are another possible solution to overloads [13]. A mobile agent is a program that is able to migrate from host to host on a network under its own control. The agent chooses when and where it will migrate and interrupts its own execution and continues elsewhere on the network. Every agent therefore needs to keep track of variations in load. In contrast to this, load variations in TRACE are handled by the RMs, and all other agents solely carry out domain problem solving activity. This results in reduced reorganization overhead.

Agent cloning is proposed as a more general approach than agent mobility [4]. Cloning is the act of creating and activating a clone agent to execute some of the agent's tasks, and is performed when an agent predicts an overload. Cloning subsumes agent mobility. Agent migration can be implemented by creating a clone on a remote machine, transferring the task from the original agent to the clone and dying. Thus agent mobility is an instance of agent cloning. Agent mobility and cloning do not ensure fairness of resource allocation. TRACE allocates resources to tasks on the basis of their criticality. Thus if the system is overloaded, low priority tasks are decommitted to accommodate high priority ones. This allows the performance of the system to degrade gracefully.

There are several interesting extensions possible for future work. Currently, the reorganization cycle time in TRACE is fixed. Finding a means of dynamically varying the reorganization cycle time will make the framework more adaptive. We are also working towards comparing TRACE with existing approaches.

6. REFERENCES

[1] S. Cammarata, D. McArthur, and R. Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, Karlsruhe, Federal Republic of Germany, 1983.

[2] D. D. Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks.* PhD thesis, University of Massachusetts, February 1983.

[3] K. Decker and V. Lesser. Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 73–80, San Francisco, CA, June 1995.

[4] K. S. Decker, K. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In C. Zhang and D. Lukose, editors, *Multi Agent Systems*, pages 63–75. Springer-Verlag: Berlin, Germany, 1997.

[5] M. d'Inverno, M. Luck, and M. Wooldridge. Cooperation structures. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 600–605, Nagoya, Japan, 1997.

[6] E. H. Durfee, D. L. Kiskis, and W. P. Birmingham. The agent architecture of the University of Michigan digital library. *IEE Proceedings on Software Engineering*, 144(1):61–71, February 1997.

[7] S. Fatima. *An Adaptive Organizational Policy for Multi-Agent Systems.* PhD thesis, Department of Computer and Information Science, University of Hyderabad, India, December 1999.

[8] S. Fatima and G. Uma. AASMAN: An adaptive organizational policy for a society of market based agents. *Sadhana Academy Proceedings in Engineering Science*, 23(4):377–392, August 1998.

[9] S. Fatima and G. Uma. An adaptive organizational policy for multi agent systems — AASMAN. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 120–127, Paris, France, 1998.

[10] F. Guichard and J. Ayel. Logical reorganization of DAI systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 118–128. Springer-Verlag: Berlin, Germany, January 1995.

[11] T. Ishida, L. Gasser, and M. Yokoo. Organization self design of production systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):123–134, April 1992.

[12] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley: Reading, MA, 1994.

[13] V. A. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, pages 26–37, July 1998.

[14] R. G. Smith. *A Framework for Distributed Problem Solving.* UMI Research Press, 1980.

[15] W. Walsh and M. Weldon. A market protocol for decentralized task allocation. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 325–332, Paris, France, 1998.

[16] Michael P. Wellman, W. P. Birmingham, and E. H. Durfee. The digital library as a community of information agents. *IEEE Expert*, 11(3):10–11, 1996.

[17] F. Ygge and H. Akkermans. On resource oriented multi commodity market computation. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 365–371, Paris, France, 1998.