17. A Market-Based Model for Resource Allocation in Agent Systems

Jonathan Bredin[1], David Kotz[1], Daniela Rus[1], Rajiv T. Maheswaran[2], Çagri Imer[2], and Tamer Başar[2]

[1] Department of Computer Science, Dartmouth College, 6211 Sudikoff Laboratory, Hanover, NH 03755-3510, USA
{jonathan,dfk,rus}@cs.dartmouth.edu
[2] Coordinated Science Laboratory, University of Illinois
1308 West Main Street Urbana, Illinois 61801, USA
maheswar@uiuc.edu,{imer, tbasar}@control.csl.uiuc.edu

Summary.

In traditional computational systems, resource owners have no incentive to subject themselves to additional risk and congestion associated with providing service to arbitrary agents, but there are applications that benefit from open environments. We argue for the use of markets to regulate agent systems. With market mechanisms, agents have the abilities to assess the cost of their actions, behave responsibly, and coordinate their resource usage both temporally and spatially.

We discuss our market structure and mechanisms we have developed to foster secure exchange between agents and hosts. Additionally, we believe that certain agent applications encourage repeated interactions that benefit both agents and hosts, giving further reason for hosts to fairly accommodate agents. We apply our ideas to create a resource-allocation policy for mobile-agent systems, from which we derive an algorithm for a mobile agent to plan its expenditure and travel. With perfect information, the algorithm guarantees the agent's optimal completion time.

We relax the assumptions underlying our algorithm design and simulate our planning algorithm and allocation policy to show that the policy prioritizes agents by endowment, handles bursty workloads, adapts to situations where network resources are overextended, and that delaying agents' actions does not catastrophically affect agents' performance.

17.1 Introduction

Agents are clean abstractions for constructing multiple-user applications. In particular, the abstraction is useful for networked applications where agents represent competing or cooperating principals. We believe that using the agent model, it is possible to quickly construct openly networked architectures where computational resources are distributed around the network for remote and visiting agents to use. There is, however, little incentive for service providers (hosts) to accommodate arbitrary agents. Hosts expose themselves to additional resource contention and risk inherent in offering any additional network service. Conversely, agents have no incentive for responsible resource usage nor mechanisms to assess the costs of their actions.

We solve these problems by having agents use electronic currency to purchase the computational resources that they use. The currency's value gives hosts incentive to entertain arbitrary agents, finite budgets bound agents' impact on the network, prices convey the cost of an agent's actions, and market competition serves as a primitive coordination mechanism.

We argue in Section 17.2 that markets solve problems at many levels faced by agent systems. In Section 17.3, we summarize work we have done to provide a secure marketplace for mobile agents. We provide a resource-allocation policy in Section 17.4 on which we derive an algorithm to minimize a mobile agent's execution time of a sequence of tasks given a budget. Section 17.5 presents simulation results of agents visiting hosts using our resource-allocation policy. We show that it effectively prioritizes agents according to endowment and that our planning algorithm's performance degrades gracefully as network delay increases. We summarize relevant related work in Section 17.6 and conclude by fleshing out our future research direction and describing the applicability of our results to more general agent systems in Section 17.7.

17.2 Markets

We promote agent architectures where agents' potentials are represented through a common parameter– their endowments. Agents may spend their endowments how they see fit to optimize their performance. Rather than specify absolute rules, hosts specify policies to guide agents. While agents may deviate from these policies to tune their performance, they do so at the expense of depleting their endowments, and possibly their lifetimes. The architecture type we propose is that of a market, though the extent of market implementation may vary. In this section, we give a high-level description of market structures applied to agent systems.

Our computational markets involve agents purchasing the computational resources that they consume from hosts. Agents can sell their services to users and other agents. Hosts accumulate revenues that they redistribute to their users, who use the currency to launch their own agents. Figure 17.1 sketches the exchange of money in our market.

Currency exchange can happen at many levels to provide a cleaner design, fault tolerance, and incentive mechanisms. At the most basic form, an agent's currency represents its potential to act in the network. Currency exchange can be modeled at the design level to provide a useful architectural tool by quantifying the limits of agents' actions.

More involved markets have agents and hosts exchange cryptographically verifiable electronic cash. Electronic cash serves as a security mechanism and verifies the spender. This type of budget constraint bounds the havoc that a malicious agent can wreak and provides a limited form of fault tolerance.

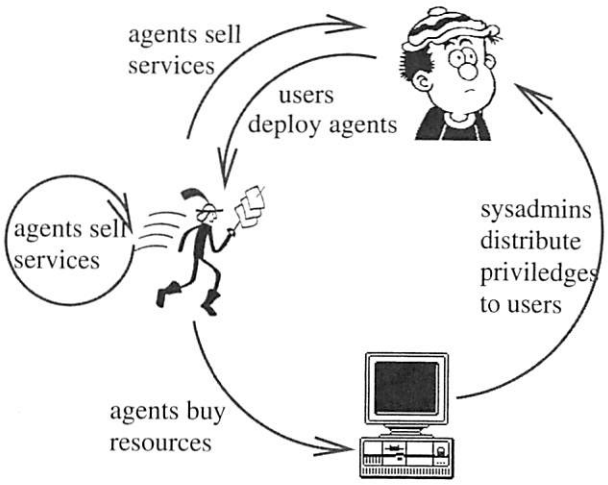

Fig. 17.1. The flow of money in our computational market.

In a still stronger market, participants exchange virtual currency that is redeemable for legal tender currency (e.g, U.S. dollars). Here, the currency verification process is more important and requires more scrutiny. The benefit, however, is that administrative domains become flexible. Resource owners may lease their equipment to agents outside their domain, recouping the cost of capital. Conversely, system administrators may temporarily expand their domains when the need for extra computation arises by buying outside resources.

In each of these examples, we rely on a pricing system. Prices signal the costs of agents' actions and allow agents to plan accordingly. When agents crowd resources, resource prices rise and deter further congestion. Thus, prices serve to balance computational load over time and through the network [620, 149].

17.3 Secure Transactions

The first step towards establishing any market is to promote methods of secure exchange. The most obvious part of secure transactions is a verifiable currency. Sometimes, exchange requires more than just valuable currency and we have implemented a protocol for a trusted third party to mediate transactions between agents who may not trust one another. Agents must be able to find and negotiate with vendors of computational resources, and we have implemented an architecture for agents to locate machine resources through a network of resource management agents. Finally, agents are ultimately subject to the wishes of their hosts, but we argue that interaction between hosts and agents is beneficial to both and that repeated interaction

is likely. Repeated interaction and exchange foster cooperation between hosts and agents.

17.3.1 Currency

There are many existing electronic currency systems [276, 363, 449, 500]. As a proof of concept, we have implemented electronic currency and a hierarchy of banks in Agent Tcl, a mobile-agent system [95]. Our currency is verified using PGP and incurs a lot of overhead in comparison to other electronic currency systems, but clearly any of the existing more efficient mechanism could also be used. Most mobile-agent systems cryptographically verify the identity of agents, so verifying currency on agents' arrival does not pose significant additional overhead. After agents verify their currency at a host, they can convert a global currency to local "script" similar to what is done in the Millicent electronic currency system [276]. This local verification reduces transaction costs enough to allow small efficient transactions.

17.3.2 Repeated Interaction

It is frequently the case that repeated interaction among agents creates incentive for cooperation. In this subsection we focus on the effects of frequent interaction between *mobile* agents and their hosts. In this context, a mobile agent is a computational process that may autonomously relocate its execution from one host to another.

Currency validation protects sellers, specifically mobile-agent hosts. Protecting agents from agents and hosts from agents appear to be tractable problems and there has been much progress in the area. Protecting mobile agents from their hosts is difficult, however. A host provides an agent with an execution environment and hence, the mobile agent is at the mercy of its host. There is little to prevent the possibility of the host robbing visiting mobile agents.

While it may always be possible for hosts to molest mobile agents, we believe that it is in hosts' best interests to provide safe and reliable execution environments. In establishing a market for computation, it must be the case that transactions benefit both hosts as well as agents. If this were not the case, then either agents or hosts would not participate in the exchange.

Additionally, we believe that mobile-agent applications will likely rely on frequently sending small agents to remote sites. Thus, a mobile-agent market for computational resources will see repeated interaction between hosts and agents. Since molesting an agent risks upsetting a host's income stream, there is incentive for the host and agents to cooperate. Axelrod sees many examples of cooperation development in scenarios where there is repeated interaction among human agents and that often little communication is necessary between agents to foster cooperation [46].

17.3.3 Arbiter

While we believe that in many circumstances an agent will deal with known agents and facilities, it is likely that at some point it will be necessary to conduct transactions between untrusting parties. There may be little reason to believe that there will be future transactions, or agents may prefer to conceal their identities. To support such transactions, we have implemented an arbiter protocol for use within Agent Tcl [95].

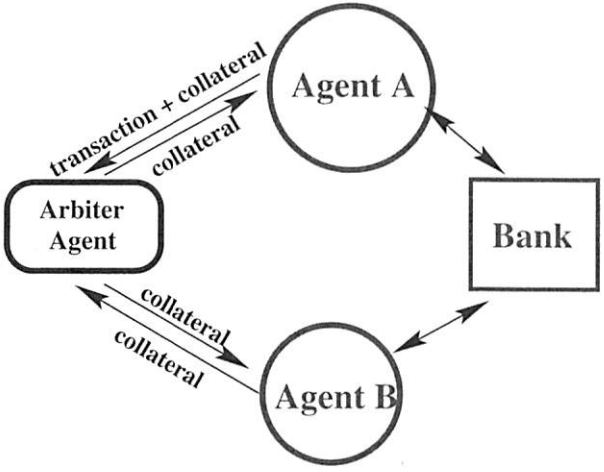

Fig. 17.2. The arbiter protocol for secure transaction between agents. Agents A and B send collateral and the transaction to a trusted third party, the arbiter agent. Once the arbiter agent receives payments from the agents, it delivers the product and payment to the agents. After an agreed upon time, if neither agent complains, the arbiter returns the collateral to agents.

In our arbiter protocol, agents arrange for a trusted third party, an arbiter, to conduct the transaction. The arbiter asks each agent for the information to be exchanged between the agents, and an amount of collateral equal to the value of the transaction from each agent. If, by an agreed upon time, neither agent is unhappy with the resulting transaction, the arbiter returns the collateral to the agents. If, however, one agent does not receive the goods or payment that it expected, the arbiter retains both agents' collateral until the transaction can be policed.

17.3.4 Resource Managers

To facilitate resource location and access negotiation in the D'Agents mobile-agent system, every host machine has a set of well-known resource-management agents [286]. These agents are responsible for allocating their respective resources and all resource consumption requests are directed to them.

Resource managers are agents and use the same communication protocols as every other agent, so transactions between an agent and a resource manager do not require any additional protocol. Any agent request to use a resource is automatically forwarded to the relevant resource manager, however, and resource consumption can be transparent to consumer agents.

Resource managers allow system administrators to tune resource access to fit individual resource consumption habits. Certain resources may require specialized policies or coordination with other resources. Several resource managers may work together to prevent deadlock or to track suspicious resource usage.

17.4 Allocation Mechanism

We now provide a specific market model that for use in allocating computation to mobile agents. For a more rigorous system formulation, we refer the reader to [97, 96]. In our model, a mobile agent attempts to minimize the time taken to execute a series of tasks each requiring consumption of a particular service.

For example, an agent may wish to retrieve an image at one site, process the image using a computationally intensive algorithm at another, and then deliver the image to be displayed at a display located near its user. Figure 17.3 depicts the agent's example itinerary. Each of the three sites that the agent visits may allow the agent to pay more to complete its task faster.

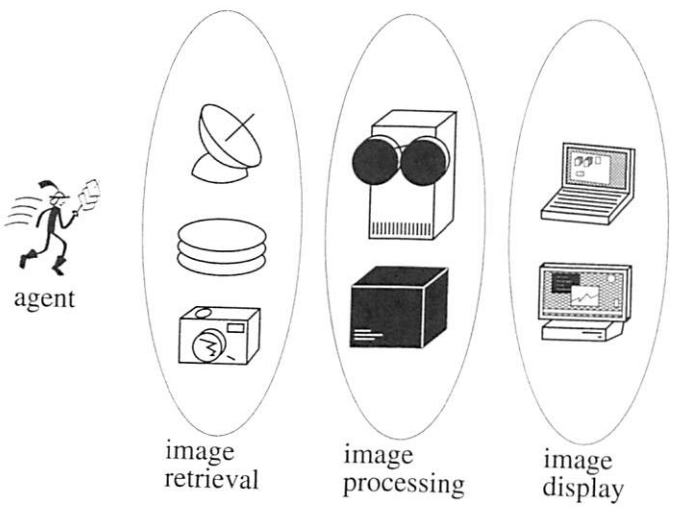

Fig. 17.3. An example of a mobile agent's itinerary. The agent must visit one host in each group and choose at what priority to execute at each host visited.

After an agent finishes all execution, its remaining endowment vanishes. Each host offers one of K computational services to the set of agents at the site. To extract agent demand, hosts solicit bid functions from the agents. The bids represent rates at which the bidder will pay for access to the service. The i-th agent receives service at a rate equal to the capability, c_k^i, of the k-th host times the agent's bid, u_k^i, relative to the sum of all bids the host receives, $\theta_k = \sum_i u_k^i$. So the i-th agent computes its task at site k at a rate of:

$$v_k^i = c_k^i \left(\frac{u_k^i}{u_k^i + \theta_k^{-i}} \right) \tag{17.1}$$

where $\theta_k^{-i} = \theta_k - u_k^i$ represents the sum of competing agents' bids. The amount of time an agent spends to complete its task is the size of the job, q_k^i, divided by the computational rate, v_k^i, which is:

$$t_k^i = \frac{q_k^i (u_k^i + \theta_k^{-i})}{c_k^i u_k^i} \tag{17.2}$$

The amount of currency an agent spends is its bid, u_k^i, multiplied by the time taken, t_k^i, is:

$$e_k^i = \frac{q_k^i (u_k^i + \theta_k^{-i})}{c_k^i}. \tag{17.3}$$

We derive a bidding strategy that minimizes the time to complete an agent's itinerary given knowledge of the itinerary's task sizes, the amounts competing agents bid, the capacities of all hosts to be visited, a fixed budget constraint, and the assumption that the agent has no need for currency other than to complete its set of tasks at hand.

We begin by assuming that the agent's bid is small enough that competing agents will not change their bids significantly in response to the agent's bid. From this assumption, we derive $f^i(\theta_1^{-i})$, the i-th agent's optimal bid at the first host as a function of competing agents' bids, θ_1^{-i}. The agent's naive bidding strategy for the i-th is:

$$u^i = f^i(\theta_1^{-i}) = \max \left(\frac{\alpha^i - \beta^i \theta_1^{-i}}{\beta^i + \frac{\gamma^i}{\sqrt{\theta_1^{-i}}}}, 0 \right). \tag{17.4}$$

The parameters α^i, β^i, and γ^i describe the i-th agent's itinerary and the state of the hosts it will visit:

$$\alpha^i = I - \sum_{k \neq 1} \frac{q_k^i}{c_k^i} \theta_k^{-i} \tag{17.5}$$

$$\beta^i = \frac{q_1^i}{c_1^i} \tag{17.6}$$

$$\gamma^i = \sum_{k \neq 1} \frac{q_k^i}{c_k^i} \sqrt{\theta_k^{-i}} \tag{17.7}$$

where I is the agent's remaining endowment.

In reality, however, other agents will change their bids in response to the agent's bid. We must augment our bidding strategy to accommodate other agents' responses by finding an equilibrium bidding level at which all agents' bidding functions are satisfied. We can find the equilibrium by transforming each agent's bidding function domain to a domain common to each agent. The original domain operates over the over the space of the sum of all other agents' bids, θ_1^{-i}, while the transformed domain, θ_1, includes the bid of the i-th agent. The result is:

$$u^i = g(\theta_1) = \frac{(\alpha^i - \beta^i \theta_1)^2}{2\gamma^{i2}} \left(-1 + \sqrt{1 + \frac{4\gamma^{i2}\theta_1}{(\alpha^i - \beta^i \theta_1)^2}} \right) \tag{17.8}$$

when $\theta \in (0, \alpha^i/\beta^i)$ and $u^i = g(\theta_1) = 0$ otherwise. The fraction α^i/β^i represents the rate at which the agent can pay to compute at the first host. If α^i/β^i is less than θ_1, the agent does not have enough money to be serviced given the current level of site congestion. If the quotient is negative, then the agent must wait until congestion at future hops subsides. To ensure that $g(\theta)$ is continuous, we require γ^i to be strictly positive. Normally, γ^i would only be zero when the agent bids for its last task. We approximate the situation by assigning γ^i an arbitrarily small positive value to keep $g(\theta)$ continuous.

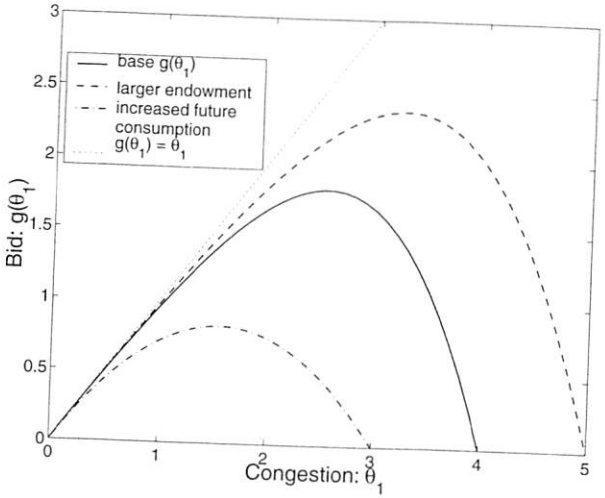

Fig. 17.4. The form of an agent's bid as a function of all bids submitted to the server. As the agent's endowment increases, the function scales outward. Increasing the size of future tasks results in a smaller gentler curve.

Figure 17.4 sketches three examples of agents' bidding functions for the next task. The function, $g(\theta)$, has a few important properties. First, a positive bid computed with $g(\theta)$ guarantees that the agent can complete the task at the current level of congestion. After completing the task with the bid computed from $g(\theta)$, the agent will have enough cash remaining to finish its itinerary given current network conditions. The function is continuous and concave in the upper right quadrant. Finally, $du/d\theta$ equals one at the origin.

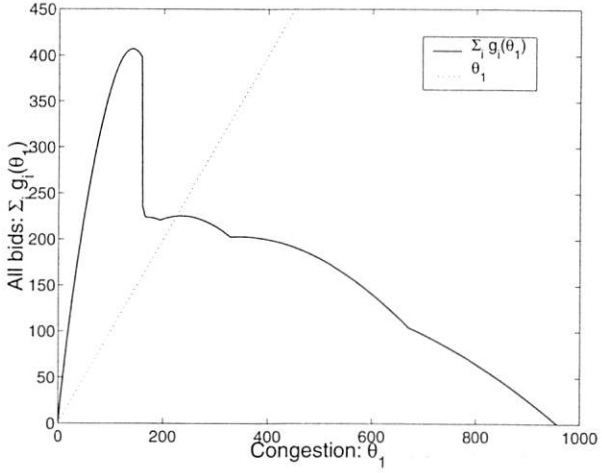

Fig. 17.5. Sample plot of the sum of 16 agents' bids a function of the sum of all bids at the server. Equilibrium occurs at the intersection of the dotted line and the plotted curve, where no agent wishes to change its bid.

We complete the description of our resource-allocation policy by allowing each agent to submit the parameters α^i, β^i, and γ^i that describe its itineraries, wealth, and perceived state of the network. If there are multiple agents visiting, the host constructs all the bidding functions, sums them, and finds a value of θ at which $\theta = \sum_i g(\theta)$ and allocates a fraction of the computational resources to the i-th agent equal to $g^i(\theta_1)/\theta_1$. This determines a Nash equilibrium for the underlying non-cooperative game. If there is only one agent, then there is no contention for the resource and the host assigns the agent the full share of resources. Figure 17.5 demonstrates an example of 16 agents' bidding function and the resulting equilibrium, which we show as the intersection of the sum of the individual bidding curves and the 45° line. In [97], we prove that when there are two or more agents at a server, there is exactly one bidding level at which agents will be satisfied with their bids given their bidding functions. This means that the underlying game has a unique Nash equilibrium.

Note that to utilize Equation 17.8, an agent must already have formulated a route. If we fix the costs and times required for an agent to complete its

tasks at every host, the problem of choosing the hosts to visit to minimize execution time under its budget constraint is NP-complete. The problem is the constrained shortest path problem to which the knapsack problem can be reduced [15, 96]. We are currently working on algorithms that use dynamic programming to approximate optimal routing with performance guarantees.

As a reasonable heuristic, the i-th agent can use Equation 17.8, by setting the values of θ_k^i and c_k^i parameters for tasks after the present task, to represent the mean values of hosts that may complete the agent's k-th job. We demonstrate the effectiveness of the heuristic in the next section.

17.5 Simulation

We generate a network topology with the GT-ITM stochastic network topology generator [121] to implement a simulation of a network of mobile-agent hosts in the Swarm simulation system [374]. In GT-ITM, a network consists of a hierarchal system of transit domains connecting stub domains. The networks in our simulation have 100 host nodes in three levels of transit domains. Because we focus on expenditure planning, we choose network delays that do not dominate job execution times.

Host capacity is determined by a positively truncated Gaussian random variable with a positive mean. Hosts offer one of eight services to mobile agents that are created at a Poisson rate uniformly across the network.

At creation, we give each agent an itinerary of tasks and an endowment of currency with which to buy computation from hosts. The number of tasks comprising itineraries is exponentially distributed. We choose tasks uniformly from the eight services that hosts offer. In our simulations, job sizes are either exponentially or Pareto distributed. We choose the endowment to be a positively truncated random variable multiplied by the sum of the agent's task sizes. This random variable represents the agent's owner's preference that the agent completes its itinerary quickly.

```
1:    t_min := ∞;   nextHost := ∅
2:    for all hosts k offering service next in itinerary do
3:       t_k := [transferLatency to: k from: currentHost]
            + q_k g(θ_k^{-i}) / (c_k(θ_k^{-i} + g(θ_k^{-i})))
4:       if t_min > t_k then
5:          t_min := t_k;   nextHost := k
6:       end if
7:    end for
8:    return nextHost
```

Fig. 17.6. Algorithm: Choose Next Site for Agent i

The algorithm in Figure 17.6 demonstrates how agents choose which sites to visit. For all but the next set of hosts to be visited, the agent plans to visit sites with average capacity, c_k, and congestion, θ_k, for hosts offering the k-th service. The agent assumes that there will be no change in bidding level, θ_k^{-i}, and chooses the next site to be the one that minimizes the sum of execution times and network transfer times for the next hop. Thus, our routing algorithm is greedy and naive.

When an agent arrives at the site, it commits itself to finishing its next task at the site. The agent submits parameters α_i, β_i, and γ_i, from Equations 17.5-17.7, to describe its current task and ability to pay for service. The host uses the algorithm in Figure 17.7 to redistribute priority whenever an agent arrives or departs. The algorithm forms a bid-response function, $\sum_i g_i(\theta)$, and conducts a bisection search to find a positive value of θ at which no agent wishes to change its bid.

```
1:   while true do
2:     t := time since last arrival/departure
3:     for all agents i do
4:       deduct t g_i(θ) from agent i's endowment
5:     end for
6:     add new agent or remove departing agent
7:     for all agents i do
8:       query agent i for α, β, and γ
9:       use Equations 17.5-17.7 to build g_i(θ)
10:    end for
11:    search for θ = Σ_{i=1}^{N} g_i(θ) in (0, max_i(α_i/β_i))
12:    for all agents i do
13:      v_k^i := c_k g_i(θ)/θ
14:    end for
15:  end while
```

Fig. 17.7. Algorithm: Allocate Resources for Host k

17.5.1 Effectiveness

We would like to first verify that our resource-allocation method stratifies agents' performance in a reasonable fashion. After the network has reached a steady state, we designate seven percent of all new agents to be test agents. These test agents all share a common start host and task-type sequences, but their endowments uniformly span two standard deviations, σ, around the mean endowment, μ. We measure the performance of each test agent against what it could achieve in a network with zero resource contention. This idealized measure is the shortest path from the start host to visit hosts that offer services to complete the agent's itineraries. The edge lengths of the path are the sum of the network transfer latencies from the previous host

and the time required to process the job given that no other agent requests service, q_k/c_k.

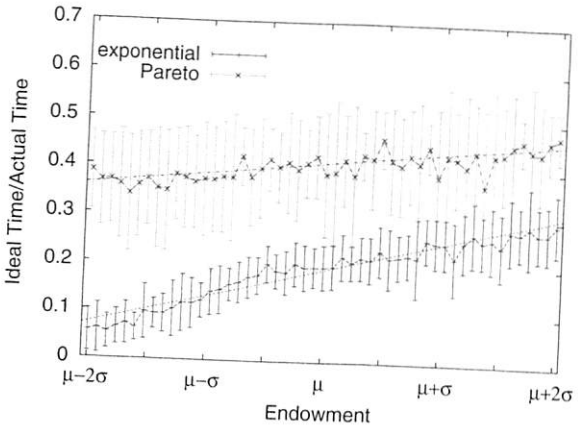

Fig. 17.8. Endowment versus ideal time relative to actual time. We plot the observed mean and standard deviations for each endowment level for two experiments for agents two standard deviations, σ, around the mean endowment, μ. One experiment has exponentially distributed job sizes and the other uses a Pareto distribution.

We run four experiments to show how agents are prioritized. There are two variables describing the experiments: workload and utilization. The two workloads are differentiated by agents' job size distribution. One workload uses an exponential distribution and the other uses a Pareto distribution.

There are two levels of utilization in the experiment. One level is approximately 70 percent of capacity while the other has agents arriving with jobs of about 140 percent of capacity. In the latter situation, it must be the case that some agents will not be able to complete their tasks.

Figures 17.8 and 17.9 show the performance of agents with different endowments at 70 and 140 percent utilization, respectively. We plot the means as well as the standard deviations for each endowment level. The Pareto distribution has a much lower median than an exponential distribution with the same expected value. Hence there are more small jobs that are easier to schedule and agents with Pareto distributed jobs generally perform their itineraries more quickly than those with exponentially distributed job sizes.

In the experiments run at 70 percent utilization, there is a weak linear relationship between agents' expenditure and their performance. Since the system can accommodate all agents, there is little need to discriminate against agents with low endowments to improve the performance of agents with larger endowments.

When agent demand exceeds system capacity, however, hosts must ignore poorer agents to allow richer agents to complete their tasks. In this scenario,

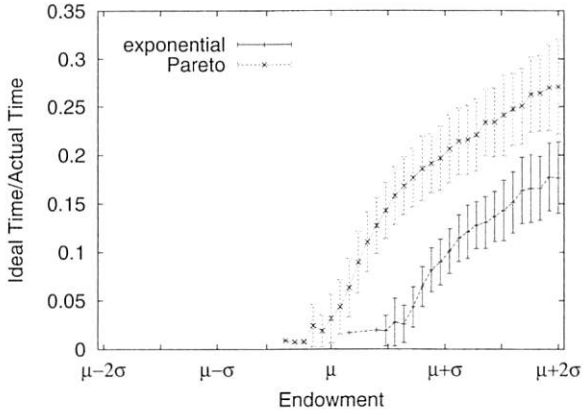

Fig. 17.9. Endowment versus ideal time relative to actual time when agent requests exceed capacity. We plot observations two standard deviations, σ, around the mean endowment μ.

there is a strong relationship between agents' endowment and performance and agents with endowments below the mean do not complete their jobs.

17.5.2 Network Delay

One criticism of market systems is the belief that they are sensitive to the accuracy of knowledge concerning the state of the world. We test this belief by varying the latency incurred by agents jumping from one host to another. Agents have accurate information concerning the current state of the network, but increasing the transfer latency delays agents' actions and simulates agents using aged information.

Figure 17.10 demonstrates that agents' performance decays gradually as the quality of their network information decreases. The figure plots network delay compared to a reference network with the performance that agents receive in the reference network. We observe that increasing network delays does not cripple our resource-allocation or planning algorithms.

17.5.3 Price Structure

Figure 17.11 shows a histogram of the logarithm of price of computation at a server in our simulation. We observe that price is log-normally distributed over time. With a log-normal distribution, agents can expect that price will be stable for the most part, punctuated with intense, but brief, periods of high prices.

These periods of contention aggravate agents' ability to complete their tasks. When the price of computing is high, it is frequently the case that

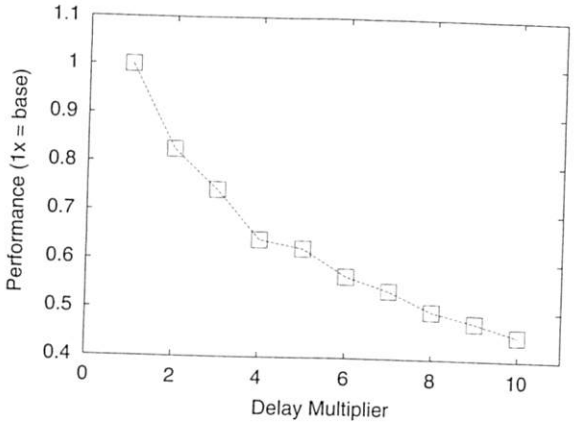

Fig. 17.10. The mean performance of agents over all endowment levels versus network delays relative to the mean performance of agents operating in a network with a delay multiplier of one.

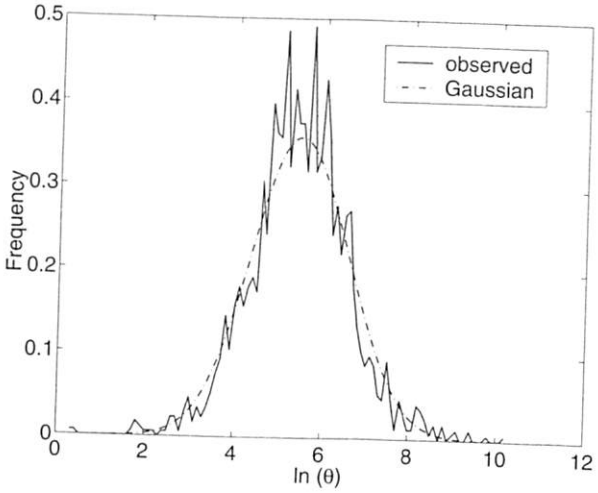

Fig. 17.11. A histogram of the logarithm of price at a host over time. We observe that price is roughly log-normally distributed, resulting in intense but brief periods of high prices.

one agent is consuming the majority of the host's computational resources to complete a small task. While the price is high, other agents' tasks are put on hold, but the flow of agents into the system continues. So it is imaginable that the host's load will not fall back to its initial level for an extended period. These periods add to the variability of agents' performance and in Section 17.7 we will discuss methods for agents to trade risk to decrease their performance volatility.

17.6 Related Work

Economic ideas for controlling computational resources are not new; in the sixties, Sutherland established auctions to schedule computer time among users [574]. Spawn is perhaps the most cited work dealing with computational economic systems [620]. In Spawn, agents participate in auctions to buy processor time to run computationally intensive jobs. The pricing system pairs idle processors with jobs and improves utilization in distributed systems. Clearwater *et al.* use double auctions to allow agents to trade climate-control resources within an office building [178]. The result is that climate control resources are more effectively allocated with energy saving of up to ten percent.

Forms of market-based control have been a part of mobile-agent systems from the field's beginnings. One of the first mobile-agent systems, Telescript [631], supports a fault-tolerance and security measure where agents carry "permits" to access specific resources. A permit's power diminishes over an agent's lifetime, thus limiting the agent's lifetime. A permit for one resource is not easily converted to another resource permit. A more general policy would be for hosts to issue a common permit in the form of a verifiable electronic currency.

The Geneva Messengers project [605] applies market ideas to allocate CPU usage and memory to visiting "messengers," lightweight mobile programs implemented in a Postscript-like language. Host sites heuristically set prices by examining the amount of resources requested by the present messengers.

POPCORN [508] is a system for distributing "computelets" to hosts on the network. POPCORN assigns computelets to anonymous entrepreneurial hosts through a double auction. Because agents do not choose their sites and no mobility after initial placement is allowed, there is no reason for agents to plan their expenditures. Additionally, the framework does not permit intercomputelet communication. POPCORN is ideal for parallel computation intensive programs where interaction among threads is limited. Our system is more general, but mobile agents must weigh many factors in choosing their hosts and agents are accountable for planning their routes and expenditures.

Boutilier *et al.* solve sequential resource-allocation problems [87]. It is difficult to construct mechanisms where it is rational for buyers to truthfully reveal their preferences, but Boutilier *et al.* resolve the problem by ensuring that all users vying for resources interact with identical agents. They prove that it is rational for users to express their preferences to their agents, who then compete in iterated sequential auctions until a stable resource allotment is found. The method can handle many traditionally difficult assignment problems where goods may be complements or substitutes to one another. The technique is centralized and more general than what we model in that it makes no assumptions on resource values and relations, but it is also computationally more expensive.

Tatonnement is another centralized resource-allocation method where buyers iteratively adjust purchase amounts in response to sellers' changing prices. The WALRAS algorithm [152] is a system for finding equilibrium where participants have convex utility functions and there is gross substitutability among goods (goods are not complementarities for one another), but often converges to a solution even when gross substitutability is violated.

Market-based resource control appears in non-research applications. In 1997, *distributed.net* [376] entered a contest held by RSA Labs to break a 56-bit secret-key encrypted message. The project wrote a client that users downloaded to search portions of the key space on users' computers. The winning user, or team of users, received a prize from *distributed.net*. The prize is a probabilistic payment for users' computing resources by *distributed.net*.

Another use of market-based control is Enron Communications' [341] establishment of bandwidth trade among its customers. Enron Communications' customers can resell their unused bandwidth to other customers in an automated fashion. Currently, trade is restricted to bandwidth of a single link connecting New York and Los Angeles.

17.7 Conclusions

We provide the framework upon which to build market-based resources in the D'Agent mobile-agent system. This framework includes a secure currency, well-known resource manager agents, arbiter agents for larger transactions, and an environment with repeated interaction to foster cooperation among agents and their hosts. We provide a resource-allocation method for computational priority and derive an algorithm that allows a mobile agent to plan execution of a sequence of tasks to minimize execution time given a budget constraint.

It is possible to supplement our serial bidding algorithms with other algorithms to minimize parallel task execution time. An extension would likely accompany more complex task planning as well. Our model applies to general agent uses, not just mobile ones, as our policy and algorithms minimize execution time for sequential tasks. Using our policy, it is possible to allocate any divisible resources. For example, we can apply our algorithm as a decentralized method of flow control for network bandwidth by auctioning off bandwidth.

We simulate our allocation mechanism and agents using our planning algorithms to show that our system handles different workloads and adapts to handle situations where resources are over-constrained by ignoring lower-priority agents' requests. Additionally, our system degrades gracefully as network latency increases and suggests that agents can reason effectively without complete network state information.

We observe that prices in our computational markets can be volatile, however. This volatility adds uncertainty to agents' performance and it is de-

sirable for users to have the knowledge of how their agents will perform when agents are launched. For this reason, we are currently researching reservation systems where hosts issue call options to agents. A call option gives an agent the right, but not the obligation, to buy computation at a specified time and price in the future. There will still be volatility in agents' performance, but the uncertainty associated with an agent's performance will be eliminated once the agent holds options for its computation.

Another limitation with our work is that we assume that we can parameterize every application through controlling access to a single good. In reality, the good that we control is a bundle of resources that we label "computational priority." If agent applications have similar resource usage habits, controlling the bundled resources will suffice to regulate the system. If the application space is more diverse, then it may be more efficient to separately allocate multiple resources among agents. For example, some agents may wish to purchase high-quality network connections, but not consume very much processor time, while there may be other agents with opposite needs.

Traditionally, multiple resource allotment has been a computationally difficult problem. The problem becomes intractable when an agent may substitute one good for another. Recently, however, there has been much effort spent towards allocating many resources to agents [87, 254, 484, 532]. So far, there are no bounds on the amount of time necessary to compute an allotment, but these recent projects investigate allocating numbers of goods ranging from ten to hundreds. We imagine that allocating half a dozen resources to agents will provide an efficient allocation and it may be reasonable to allocate multiple resources to agents.

Acknowledgements

This work is supported in part by Department of Defense contract MURI F49620-97-1-0382, DARPA contract F30602-98-2-0107, and an ARO/EPRI contract. We would like to thank Robert Gray for implementing the core D'Agents system and Joe Edelman for implementing the banking, arbiter, and currency systems inside Agent Tcl.