Market-based Resource Control for Mobile Agents

Jonathan Bredin, David Kotz, and Daniela Rus
Department of Computer Science
Dartmouth College
Hanover, NH 03755
rus@cs.dartmouth.edu

November 25, 1997

Abstract

Mobile agents are programs that can migrate from machine to machine in a heterogeneous, partially disconnected network. As mobile agents move across a network, they consume resources. We discuss a system for controlling the activities of mobile agents that uses electronic cash, a banking system, and a set of resource managers. We describe protocols for transactions between agents. We present fixed-pricing and dynamic-pricing policies for resources. We focus on and analyze the sealed-bid second-price auction as a mechanism for dynamic pricing.

1 Introduction

Mobile agents are programs that can migrate from machine to machine in a heterogeneous network. The program chooses when and where to migrate. It can suspend its execution at an arbitrary point, jump to another machine and resume execution on the new machine. Thus, mobile agents co-locate data and computation by bringing the computation to the data, rather than by bringing the data to the computation. Agents have the necessary autonomy to make decisions, and to interact with other agents and services to accomplish their goals. Our previous research [RGK, KGN+97] shows that mobile agents have tremendous promise as a uniform paradigm for developing distributed applications, primarily because agents are easier to write than message- or RPC-based applications, their autonomy makes them well suited to poorly connected network environments, and they remove the need for distributed applications to have their own control language.

As mobile agents move across a network, they consume resources. How can we prevent agents from over-utilizing or wasting the resources on the computers they visit?

A realistic mobile-agent system must be able to work effectively in a heterogeneous, distributed environment. In the course of its lifetime, a single agent may visit many different types of machines, which run different operating systems, and are administered by different organizations with different policies and goals. If a mobile-agent system is to work in such a situation, it is critical for the system architecture to include mechanisms for controlling resource usage, and include hooks to allow customization of the resource-control policies. Resources include everything from CPU time to screen space. The resource owners (i.e., the machines and their owners) need to control the quantity of resources used by each agent, particularly agents that are not owned by the same user or organization. It is not sufficient for each machine to control an agent's local resource usage, as might a traditional operating system; the agent's lifetime resource usage must also be controlled, to prevent agents from circulating forever. The agents themselves need to control their own resource usage, not only to optimize their own performance but also to respond effectively to limits imposed on them by the resource owners.

In this paper we present the market-based resource-control component of our mobile-agent system called D'Agents.[1] The D'Agents system supports inter-agent economic transactions with a system infrastructure consisting of a currency model, a banking system, and a set of currency-aware resource managers. This infrastructure allows (1) resource managers to set prices for resources and (2) agents mobile to dynamically adapt to the resource-pricing environment to meet their goals within their resource constraints. We propose to use sealed-bid, second-price auctions to enable the resource managers to set prices dynamically in response to changing conditions, based on a set of abstract priorities set by their human owner. We present a lower bound for the the wait time in a sealed-bid second-price auction and discuss simulation experiments.

This paper is organized as follows. After a brief discussion of related work, Section 3 gives a overview of our mobile-agent system D'Agents. Section 4 describes the currency model, the distributed banking system, the resource managers, the protocols for transactions between agents, and an application that uses them to support a traveling salesperson. In Section 5 we discuss dynamic resource pricing with auctions, analyze auctions with respect to an agent's waiting time to receive a service, and present simulation results from running auctions among agents. We wrap up with a discussion in Section 6.

2 Related Work

Mobile agents permit the migration of an agent (a computation) to a data source. This process co-location enhances programmers' flexibility in much the same manner as does multi-threaded programming, by allowing software authors another dimension to express solutions. D'Agents [Gra97, KGN+97] and Telescript [Whi94] are examples of systems supporting this form of migratory computation. Allowing processes to relocate, however, reinforces the importance of regulation. Social systems have evolved *markets* as simple distributed solutions to limiting consumption and facilitating trade.

Markets not only allow resource owners to rent out their capital for outside use, but place a limit on the lifetime of mobile processes by tying consumption to a limited currency pool. There have been several major approaches to setting prices in computational markets: sealed-bid auctions, reservation-style resource options, and priority pricing. Spawn [WHH+92] is an example of a system using second-price sealed-bid auctions to distribute resources. In Spawn, tasks compete for resources by submitting bids to the resource's owner. Bids can be expressed in a more complex manner than a simple price in systems like WALRAS [Wel96] where agents submit demand functions expressing the quantity desired at given prices. The market then computes a clearing price for goods. Waldspurger and Weihl [WW96] describe their allocation framework for selling shares that represent options for processor use. Holding a share ensures that an agent will receive a fraction of processor use

[1] Previously known as Agent Tcl.

proportional to the number of shares in circulation. Finally, it is possible to fix prices for multiple levels of service quality as described in [GSW97]. Prices can periodically be calibrated to match changing demand and encourage responsible consumption.

Markets require a mechanism to ensure reliable and legitimate transactions among agents. We propose using a trusted third party as an arbiter, though that is not the only solution. Sandholm [SL95] proposes game-theoretical situations where trade is likely to be safe given the maximum loss of a single transaction and the effect of defaulting on one's reputation. By adding a penalty for decommitting [SL96], agents can be persuaded to act in responsible fashion, while still having the flexibility to back out of a transaction in an emergency.

3 D'Agents: a Mobile Agent System

D'Agents [RGK, Gra97, KGN+97] was developed to support mobile agents written in Agent Tcl, Agent Java, and Agent Scheme (extensions of Tcl, Java, and Scheme, respectively) over the past two years. The primary goal of the project has been to implement a computational paradigm that co-locates data and computation by bringing the computation to the data (rather than the data to the computation). D'Agents supports this paradigm with *mobile agents*, which are programs that can migrate under their own control from machine to machine in a heterogeneous network. In other words, the program can suspend its execution at an arbitrary point, migrate to another machine, and resume execution on the new machine. Mobile agents are especially suited for applications on disconnected computers such as laptops and palmtops that require filtering through large amounts of data.

D'Agents reduces migration to a single instruction, provides simple communication among agents, and runs on generic Unix platforms. Our modified Tcl, Java, and (coming soon) Scheme interpreters can capture the internal state of an executing script (the stack, the contents of variables, etc.) at an arbitrary point. In addition, our modified interpreters provide a special set of commands that allow a program to migrate to other machines and to communicate with other migrating agents.

Thus, one of our mobile agents is simply a Tcl, Java, or Scheme program that runs in modified interpreters and uses the agent commands to roam through a network and interact with other agents. The program can decide to move to a new machine at any time. It issues the `agent_jump` command, which suspends script execution, captures and packages the internal state of the script, and sends this state image to an *agent server* on the destination machine (a special server runs on every machine to which mobile agents can be sent). The server restores the state image and the Tcl script continues execution on the new machine from the exact point at which it left off.

Our agents can communicate via message passing, stream connections, or remote procedure call [KGN+97]. An agent can use the Tk toolkit to present a graphical user interface on either its home machine or on a remote machine to which it has migrated.

D'Agents protects individual machines from malicious agents (as well as groups of machines that are under single administrative control) [Gra97, Gra96].

4 Resource Control

As mobile agents move across a network, they consume resources. How can we prevent agents from over-using or wasting the resources on the computers they visit? We developed a system for controlling the activities of mobile agents that uses electronic cash, a banking system, and a set of resource managers. In the following sections we describe the prototypes for each of these modules. These prototypes are modularized to enable easy experimentation with different resource managers, electronic cash models, and banking models.

4.1 Electronic Cash

To provide the necessary flexibility, autonomy, and decentralized control required in a distributed system of mobile agents, we developed a resource-control prototype based on a currency model. Our goal was to develop a protocol by means of which all machines and agents agree on a common currency unit (not necessarily tied to legal currency).

Agents carry with them a finite amount of currency, which they pay to resource owners for the resources they use. As a result, agents have the freedom to choose how to spend their currency on resources. They can dynamically trade off space and time, for example, once they have seen the relative cost of the necessary resources, according to their own encoded priorities. In addition, resource owners have the freedom to choose how to price their resources. A workstation owner that does not wish to see its CPU bogged down by outside agents will set a high price on its CPU time.[2] Another workstation owner may find it has surplus CPU cycles, memory pages, disk blocks, and so forth, and may choose to price its resources more competitively. Alternatively, the resource prices can be set dynamically using the policies described in Section 5.

4.2 The Banking System

We developed a distributed system of banks to manage currency (although we developed a simple protocol of our own, we plan to consider existing approaches used for Internet or smart-card commerce) using on-line protocols that allow agents to talk to their banks while executing a transaction. We have also addressed off-line protocols in the case where the buyer and seller agent are willing to trust each other through the mediation of a trusted third party. The off-line protocols are especially important when mobile agents operate on disconnected devices such as laptops.

The banking system functions as follows. There are a small number of *bank agents*, which have the authority to issue currency. The banks trust each other. (In a real system, it is likely that the banks would be run by accredited organizations and carefully regulated, much as the banks in today's real world.) Each bank has a well-known public key, and a secret private key. (We use PGP for our public-key cryptosystem.)

Every machine and application agent that needs to handle currency contracts with the bank of their choice. They need only know and trust that one bank.

In our model, all agents, machines, and resources are ultimately owned by some person or organization. Thus, the currency collected through the sale of resources accumulates in the name of the resources' owners. Those entities may then allocate currency to agents that they wish to send out for their own business.

[2]Or, of course, it can use the security mechanisms to completely bar foreign agents.

All currency is kept in *wallets*; a wallet is a set of *bills*; each bill is a unique, cryptographically signed document issued by one of the accredited banks. Each user and agent has its own wallet. Each bill is just a few bytes of information, indicating the name of the bank, the amount of the bill, the unique identifying number for that bill, and the bank's signature. The bank's signature is generated using its public key, and depends on all of the other bits in the bill. The authenticity of any bill can be quickly verified by checking the signature.

This agent banking system can be extended to include human users who might interact with the agents. Each computer user would have their own wallet, as would each organizational entity. If the machines are owned by the organization, then they would accumulate currency through the sale of their resources, and distribute currency to their personnel to satisfy their needs.

When a user creates an agent, the user gives the agent a few bills for its wallet. As it travels the network, the agent must buy all of its resources, using this currency. This includes CPU time, disk space, queries, *etc.* If the agent runs out of currency, it can execute no longer. We expect host systems will have enough charity to send the agent back to its home machine, or at least to send a death notice to the home machine.

4.3 Transactions Using Electronic Cash

Suppose agent A wishes to pay agent B for a resource or service. In the simplest case, agent A has a bill in its wallet for exactly the correct amount. It gives that bill to agent B, that is, the bits are copied from some variable in agent A to some variable in agent B. (If necessary, this transmission can be encrypted with the public key of agent B to prevent other agents from intercepting the bill.) Agent B adds the bill to its wallet, and agent A removes the bill from its wallet.

To prevent the same bill from being used twice, agent B should validate the bill when it is received. The following protocol accomplishes this. Agent B sends the bill to its bank, bank B. Bank B sends the bill to the bank that issued the bill, Bank A. Bank A validates the authenticity of the bill and records the fact that it has been used, so that it cannot be used again. Bank B issues a new bill to agent B for the same amount. Agent B is now satisfied.

If agent A does not have a combination of bills that sum to the correct amount, it can break one of its larger bills into smaller bills by sending the bill to bank A.

Note that neither of the banks need know or validate the identity of the agents, only the currency.

4.4 Arbitrated Transactions

To prevent cheating, where agent B accepts the money without providing the service, or where agent A uses the service and then refuses to pay, we add a trusted third-party arbiter to the transaction (see Figure 1). This arbiter is especially useful when the two agents are exchanging large amounts of currency, or have reason to mistrust each other. When the arbiter is involved, both agents A and B are required to send some collateral currency to the arbiter. After it receives the collateral from both agents, the arbiter notifies both agents. The agents complete their transaction. If either agent complains within the pre-agreed time, the arbiter retains the other agent's collateral. At the end of the waiting period, the arbiter returns the collateral to each agent if the other agent had not complained. It is not in the best interest of an agent to cheat the other agent, because it loses its collateral. It is also not in the best interest of an agent to complain arbitrarily, since it will not gain anything, and the other party will simply retaliate and cause the first agent to lose its collateral also. An audit trail by the arbiter allows a final (human) source for reconciliation of disputes.

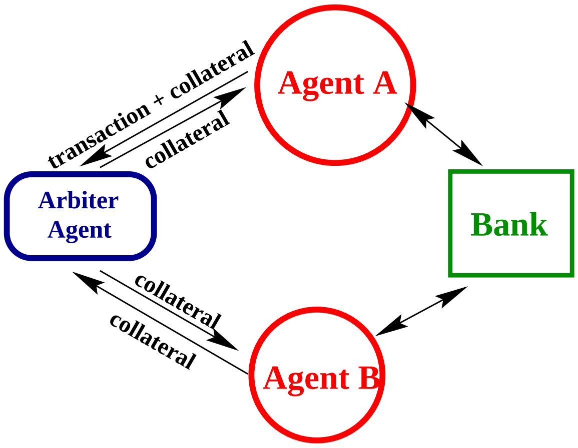

Figure 1: The arbiter protocol is used to prevent cheating. Agent A requests a service from Agent B. To start the transaction, Agent A gives the arbiter the cost of the service plus a given amount of e-cash collateral and Agent B gives the arbiter that same amount of e-cash collateral. Upon successful completion of the transaction, the Arbiter Agent returns the collateral to Agent A and Agent B.

4.5 Resource Managers

In D'Agents, nearly everything can be viewed as a resource. Some resources are closely tied to the hardware, and others represent abstract services. Most resources come in units, and most units involve time in some form. Example resources include CPU time (cycles or cycles per second of real time), main memory (byte-seconds), disk space (KByte-seconds), screen space (pixel-seconds), speaker (seconds), microphone (seconds), keyboard (seconds), network transmission (byte-seconds), database access (records searched), and so forth.

Each resource has a manager. Each *resource manager* is responsible for a particular resource (or class of resource) and fields all requests for access to that resource. Its sole job is to determine whether access should be granted, and to determine any limits on the access. For example, the CPU resource manager decides whether the agent may run at all, and if so, a limit on the total number of CPU seconds as well on the rate (virtual-time CPU second per real-time second). Once the decision has been made, it is the responsibility of the agent run-time system to enforce the decision. In addition to its duties regarding access control, the manager is responsible for setting and/or negotiating a price that it charges for the resource, and for collecting funds from the agent. So far we have used the price set by a human configuring the resource managers. In Section 5 we describe our experiments using auctions to allow the resource managers to dynamically set prices based on supply, demand, and "market conditions," such as the prices it sees offered by its "competitors".

In D'Agents we use the Safe-Tcl [BR95] infrastructure to enforce the decisions of the resource managers [Gra96]. The Safe-Tcl kernel interpreter intercepts sensitive procedure calls. On the first access, it asks the resource manager for its decision; it caches the decision to avoid asking the resource manager on subsequent resource accesses. This architecture cleanly separates the mechanism (Safe Tcl) and the policy (resource managers), allowing convenient substitution of different policies, and allowing the resource managers to remain independent of the programming language.

All of these mechanisms exist in early prototype form. We plan more experiments to measure the performance of the mechanisms, and more importantly, to design and implement pricing policies within the resource managers.

The agents circulating in this environment have the opportunity to plan their activities based on the priorities provided by their originator (such as fastest completion time, lowest price, most detailed information, and so forth), the amount of currency they can spend, and the current price of resources. An agent with a goal for fast completion time, for example, may choose an algorithm that optimizes computation time at the expense of memory usage; if the price of memory is particularly high at this site, however, it may choose to jump to an equivalent but cheaper site, or it may choose an algorithm that uses less memory but runs longer.

4.6 Application: The Traveling Salesperson

We have developed a multi-agent application that uses the banking system discussed in this paper for a traveling salesperson scenario. The premise of the application is a traveling salesperson that carries a laptop when visiting customers and uses software that helps to select vendors and products, prepare a quote, and place orders. Agents represent orders and travel to the corporation's computers where they interact with billing, inventory, and shipping agents to arrange for the purchase. Agents are also used to explore the vendor catalogs and search for products that meet the customer's needs. In all cases, the agents are mobile and can function while the salesperson's laptop is disconnected [RGK].

Figure 2 shows the structure of the application. The traveling salesperson can gather information about a particular type of purchase by sending an agent to locate all the vendors of the required type of product. The agents locate vendors by interacting with a distributed system of yellow pages, and bring back the most recent catalogs from the vendors. The catalogs are displayed as an interactive window, in which the salesperson can select items. The selected items are packaged as an order agent. This agent travels to the vendor locations and purchases the required items. This is done by paying electronic cash, using the banking system described in Section 4.2 and the arbiter protocol described in Section 4.4. When the transaction is complete, the agent returns to the salesperson's computer with the purchased items (sound clips in our prototype), which can be played locally. If the transaction cannot be completed, the agent returns to the salesperson to report on its status.

5 Dynamic Pricing

Resource manages can set the price of the resources they control by using (a) fixed-pricing strategies provided at the initialization of the system and possibly changed by the human users, or (b) by using dynamic pricing strategies that allow them to adapt to the supply and demand in the system. In this section we describe our work on dynamic pricing and illustrate it with simulation results.

Dynamic pricing is important for several reasons. First, we expect that it will be difficult to set prices manually that lead to a stable, healthy agent economy. Second, the system will change constantly, as resource supplies change (machines come and go) and resource demand changes

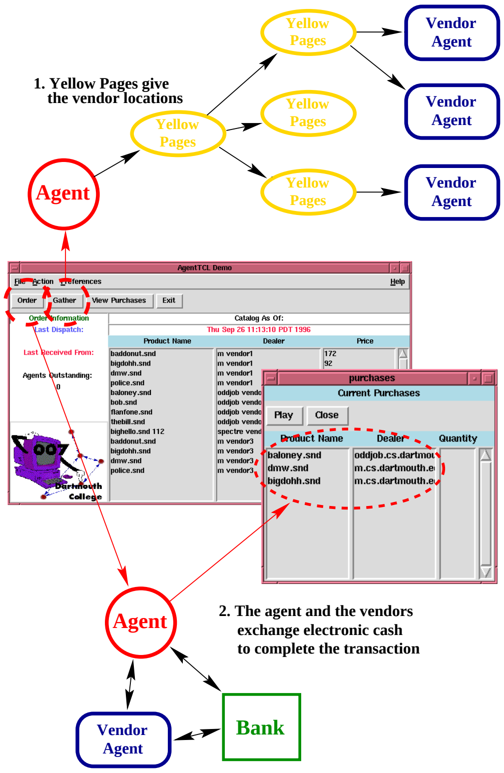

Figure 2: The architecture of the salesperson application.

(agents come and go). Indeed, there will clearly be daily and weekly cycles of activity: CPU time may be more expensive the middle of a workday than in the wee hours of Sunday morning. In addition, our experience with the World Wide Web shows that some resources are "hot" one day and not the next: e.g., a hurricane tends to make the weather-information servers busier.

5.1 Market Models for Resource Control

There are four primary approaches to dynamic pricing. Here, the seller is usually a resource manager, and the buyer is usually an agent.

1. Seller-adjusted pricing: in this approach, the seller monitors the demand for its services, adjusting the price up as demand increases and down as demand decreases. It may also be able to determine the price of competitors, and factor that into its computation. This could be a simple, ad hoc, information-poor approach. Nonetheless, it may be useful in some circumstances.

2. Negotiation: in this approach the seller and the buyer negotiate over each sale. The challenge here is to understand how the buyer, in particular, knows what prices are reasonable when it starts the negotiation. This approach could be implemented as a simple double auction like in the stock exchange.

3. Seller-driven auctions: auctions are feasible when there are many buyers competing for the same item (resource), particularly when there are more buyers than there are items.

4. Buyer-driven auctions: in these auctions, there are many sellers and relatively few buyers. The sellers broadcast their prices, essentially bidding for buyers, and the buyers choose to buy from the cheapest sellers (for example, [CMM97]).

In the following section we discuss our work on seller-driven auctions.

5.2 Dynamic Pricing with Auctions

Auctions provide a market-based mechanism for controlling resources. In auctions, the buyers are allowed to competitively set the price for goods being sold, although the seller may have a reservation price below which sale will not occur. By asking the buyers for the price, the seller needs no information about current market demand and is able to capitalize on close to all of market demand.

In this section, we explore the use of auctions as a preliminary solution for resource control in a distributed agent environment. We examine various properties of sealed-bid second-price auction markets and their benefit to multi-agent systems.

Several types of auctions can be considered to drive dynamic pricing in a distributed agent system. In the standard English auction the seller openly announces a minimal price for the good to be sold. Buyers then take turns publicly bidding or exiting the auction. Bids submitted are strictly increasing over time and sale occurs when only one potential buyer remains. Strategically, the optimal solution is for bidding to continue until the bidder willing to pay the most over-bids her competitors and pays the value of the next-closest participant.

While English auctions are efficient in extracting buyer values, they often take many rounds to complete and thus can be inefficient for agents. Instead, we use *the sealed-bid second-price auction* proposed by Vickery [Vic61]. The sealed-bid second-price auction is strategically identical to the English auction but it requires only a single round of bidding. Buyers privately submit their own valuations. The winner is the competitor submitting the highest bid, though the price of the good is the *highest losing bid.*

The best strategy for this type of auction is to submit a bid equal to the value of the good. There is no incentive to bid any lower; doing so only decreases the chance of victory without any effect on the quality of a successful auction, since the winner pays the loser's bid. Bidding higher than one's value is dangerous and risks over-payment (a fate worse than losing the auction.) Strategically, the English auction is equivalent to the sealed-bid second-price auction.

The side effect of sealed-bid second-price auctions is the near elimination of *deadweight loss* associated with competitive markets and the exclusion of potential buyers. By using only a single price, some potential buyers are excluded. Of course, it is not normally advantageous to sell below cost, but often times sellers set a price well above the real cost of their wares. Additionally, most expenses in a computational environment are *sunk.* For example, it costs the same to operate a busy CPU as it does an idle one. Conversely, some buyers would be willing to pay larger amounts than the price the seller is asking. The difference between what the buyer is willing to pay and the actual price is called *economic rent*. This potential revenue lost is known as *deadweight loss* and indicates inefficiency in the market. The rightmost shaded triangle in Figure 3 represents deadweight loss in a competitive market.

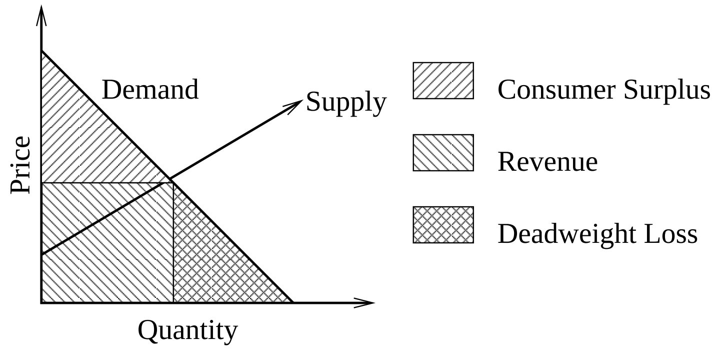

Figure 3: Simple supply and demand curves showing the utility acquired by consumers and producers as well as the loss incurred through setting a single price. The area denoted as deadweight loss shows unused demand by setting a single price while consumer surplus is the region showing how much more customers would have been willing to pay.

Since conducting a second-price auction forces buyers to pay very close to their evaluations, almost all potential revenue is collected by the seller. Only the gap in bids between the highest and second highest buyers is lost. Given enough competition, this gap should close, and the sellers' revenue increases.

Thus deadweight loss is practically eliminated through auction pricing suggesting that the market is extremely efficient in resource allocation. If resources are so scarce that there are more requests than goods, it makes sense to favor those users with higher priorities, especially when one considers that computer programs rarely feature graceful degradation and failure in one portion of execution often overwhelmingly corrupts the program's overall performance.

5.3 Modeling Auctions for Resource Allocation

To model and analyze the scope of sealed-bid second-price auctions for resource control we have developed a model and used simulation to evaluate it. In this model, many agents we will call *client agents* request service from resource managers we call *server agents*. All the servers sell a common resource called the *service*. Prices in this model are fixed with respect to quantity. Our model makes two assumptions. The first assumption is that the demand for the service is completely inelastic with respect to quantity. That is, users are willing to pay the same price per unit regardless of quantity. This assumption makes sense if the system is to be used in an environment where all work presented to the system must be completed. The second assumption is that clients have enough electronic cash to complete their schedule of requests, but are stingy in that they wish to conserve their currency pool.

Since prices are fixed with respect to quantity, the only way to affect the price is quality. Goodness of service is measured by computing the price/performance ratio. Performance, in this model, is calculated as the time elapsed between the service request and service completion, normalized for the quantity requested. In this model, every client has an expectation of performance with respect to per unit prices based on previous experiences.

A measure that connects price and quality of service allows the clients to trade expenditure for performance, but further specification is necessary for users to be able to evaluate a good. In our model, clients have threshold for the level of service for which they are willing to pay. This threshold is described for every client as an isoquant curve in the price performance space — a curve upon which every point is equally desirable. Figure 4 shows an example of such a threshold. The threshold is used by clients to determine what is an acceptable price for a service request.

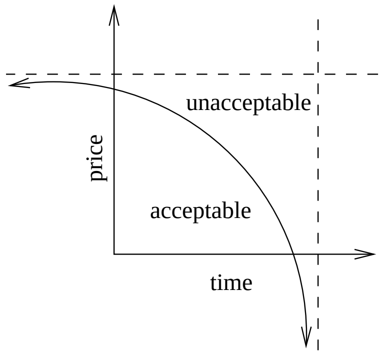

Figure 4: A sample curve describing a client's preference regarding price and performance.

5.4 Analyzing the Model

To understand the possible effect of sealed-bid second-price auctions on resource control in multi-agent systems, we compute the buyer's waiting time (average and variance) to measure the stability and distribution of wealth in the market. For simplicity, we assume that the request size is exponentially distributed and that the distribution is common to all client agents. Also, the request-arrival rates follow a Poisson distribution, using the same distribution for all client agents.

Consider for simplicity the case of one server (resource manager) and n clients (agents requesting the resource). Suppose the mean service-size request is μ and it occurs with mean frequency f. The expected wait time of the richest client is simply the server utilization times μ since the client willing to spend the most has only to wait for the user currently being serviced to complete.

The rest of the clients' expected waiting time is a bit more complex to calculate. The following results holds:

Theorem 5.1 (Lower Bounds) *In a sealed-bid second-price auction where clients have approximately the same request patterns, the lower bound for waiting time is exponential with respect to spending rank.*

Proof:

To simplify this exposition, suppose that clients' spend unique amounts with respect to each other. The analysis can be extended for the general case as well.

The variables that affect a client's waiting time are the ordinality of the client's spending compared to the competition (in other words, how many clients are willing to spend more than the current client) and the expected time for the server to service all clients with higher bids. It is possible to derive the expected wait time for any particular client given the expected time to service all the clients with higher bids.

The utilization of a M/M/1 queued server is just the Poisson ratio function:

$$\rho = R(K,z) = 1 - \frac{\frac{z^K}{K!}}{\sum_{j=0}^{K} \frac{z^j}{j!}} \tag{1}$$

[Tan95, p. 244] where K is the number of clients in the system and z is the *service ratio* (the time taken between service requests, divided by time taken up by service). So z is:

$$z = \frac{1}{f\mu} \tag{2}$$

Since the arrival times are randomly independently distributed, the chance that a client requests service when the n clients with higher bids are idle is:

$$1 - R(n, \frac{1}{f\mu}) \tag{3}$$

An approximation for the expected wait time for the n^{th} richest client follows by noting that the system is memoryless and that the probability that the server will be able to serve a client is the utilization. Integrating over time we get the expected wait time:

$$\int_0^\infty (1-\rho')\rho'^t t \; dt \tag{4}$$

Where $\rho' = R(n-1, z)$, is approximately the utilization of the queue by the $n-1$ clients with higher bids. This computation ignores the possibility that there could be a client with a lower bid being served at the time of arrival, but it serves as a lower bound on expected wait time and the approximation is more accurate for the very poor and the very rich clients.

Equation 4 integrates to:

$$\left[\frac{1-\rho'}{\ln\rho'} e^{t\ln\rho'}(t - 1/\ln\rho')\right]_0^\infty = \frac{1-\rho'}{(\ln\rho')^2} \tag{5}$$

By substituting Equation 1 into Equation 5 for ρ', we obtain the expected wait time as a function of expenditure rank, assuming that all clients have roughly the same request patterns. The resulting function for wait time is an exponential lower bound with respect to spending rank. Figure 5 plots expenditure rank versus expected wait time with request frequency 0.05 and request service time 2. Thus the service ratio, z, is 9. The wait times for the poorer clients are very severe compared to those of the wealthy because, in the estimation, the consumption of the poor have no effect on the rich. $\square$

5.5 Simulation Results

To verify the result in Theorem 5.1 we built a simulation of the sealed-bid second-price auction for one resource manager server and eight client agents. The client agents have unique preferences as assumed above. Given the spread in expected wait times modeling preferences in this manner is somewhat tricky. In congested scenarios, the poorer clients are not able to obtain service that for which they are willing to pay.

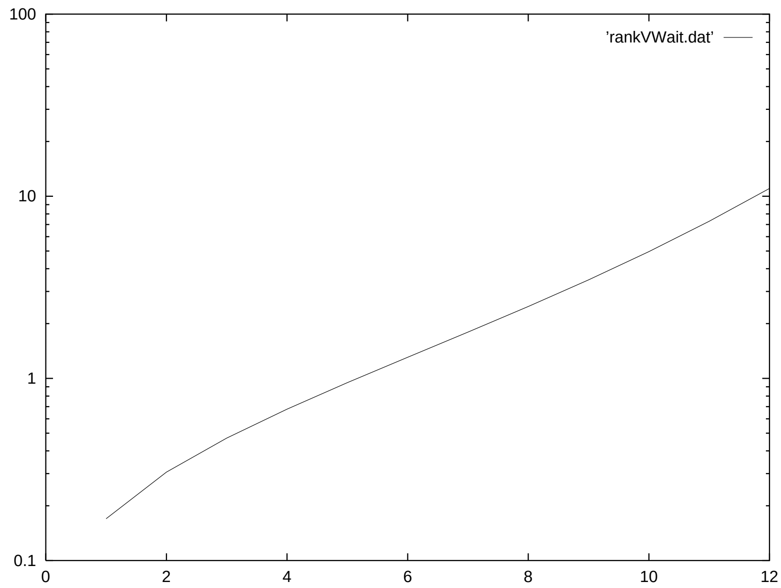

Figure 5: Plot of spending rank versus estimated expected wait time with $z = 9$ and logarithmic scale.

Congestion is an important issue for mobile agents. Intuitively, it is easy to observe that all agents using a given service are affected by congestion at all levels of spending. Richer client agents have to wait at most the current service time of a poorer clients plus the additional wait incurred by wealthier clients arriving in that time, so the estimation still serves as lower bound on expected wait time.

Figures 6 and 7 show simulation data for how spending rank determines average wait time. The plots are consistent with Theorem 5.1. We note that in this experiment, the eighth client in the scenario experienced lengthy enough wait times that caused its evaluation of service to fall negative, so it left the market.

6 Discussion

It is interesting to consider the implications of a system in which the agent currency is tied to legal currency (e.g., U.S. Dollars). Indeed, it is almost inevitable. Consider an individual that buys an expensive and powerful workstation, and hooks it into the global agent system. The hardware resources of that workstation are assets that quickly earn the individual agent currency. Thus, the legal currency spent to purchase the workstation are translated into agent currency. The user may now spend the agent currency to send agents out into the system. Or, perhaps the user can sell the agent currency to another user, perhaps a user who has only a minimal workstation and thus little potential to gain currency, in exchange for legal currency. As a result, all of the world's computational resources are sharable! Individuals can decide whether to own and maintain a workstation of their own, or to "rent" services from other machines.

The use of sealed-bid auctions is just a simple first implementation of a market system. It is unclear whether auctions provide sufficient incentive for processes to balance and smooth system wide consumption; allowing users to set their own prices gives little motivation to move given

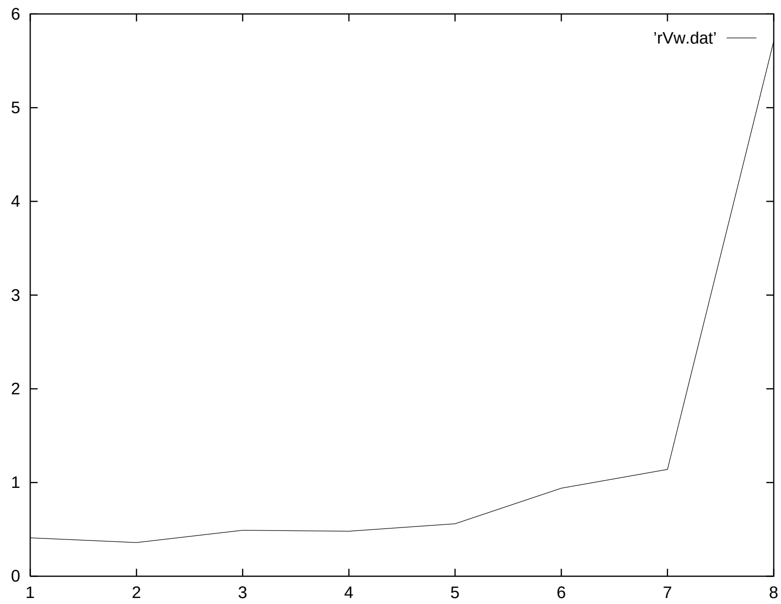

Figure 6: Plot of spending rank versus average wait time for simulation of eight clients and one server.

any consumption. However, mobile agent systems provide an enormous opportunity for modeling economic systems and games.

We will implement these and perhaps other approaches to experiment with their feasibility in the systems and applications that we envision. It is likely that a hybrid approach will be necessary: for example, to use seller-adjusted pricing when the competition is fairly light, but to switch to an auction when the demand overwhelms the supply.

Needless to say, in a real system not all of the agents and resource managers will use the same algorithm for negotiation, bidding, or price setting. We are particularly interested in the effects on an economy when different algorithms clash with each other, or (in the worst case) when a malicious or buggy agent or service uses an irrational algorithm.

Clearly, there is a tremendous amount of work necessary to understand the policies and mechanisms necessary to manage all of the resources in a complex, distributed, agent system. We believe that the currency model has tremendous potential, and that our experimental study will lead to a deeper understanding of that potential as well as concrete suggestions for implementation in a real system.

Acknowledgments

This paper describes research done in the Transportable Agents Laboratory at Dartmouth. This work is supported in part by the Navy and Air Force under contracts ONR N00014-95-1-1204, AFOSR F49620-93-1-0266, and MURI F49620-97-1-0382. Robert Gray implemented the core D'Agents system. Joe Edelman implemented the banking system and the arbiter. Joe Edelman, Mark Hoagland, and Tom Millett developed the traveling-salesperson application.

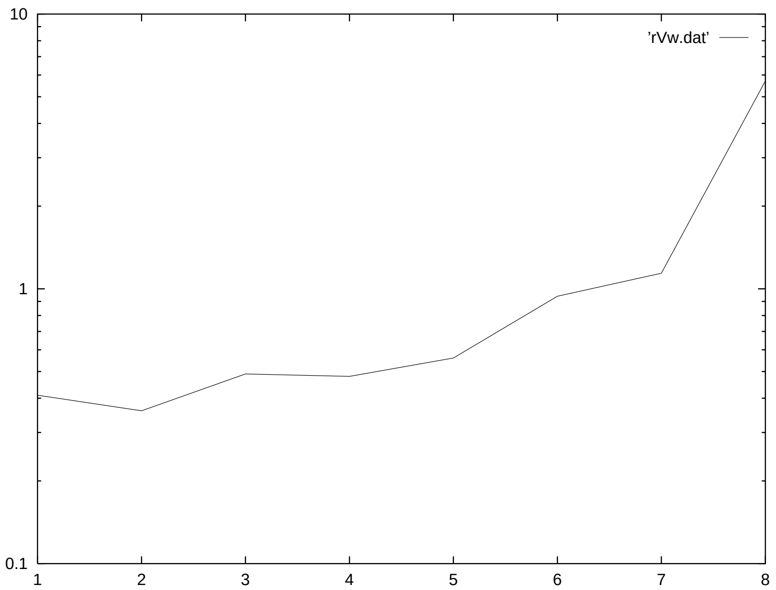

Figure 7: Plot of spending rank versus average wait time for simulation of eight clients and one server with logarithmic scale on wait time.

References

[BR95] N. S. Borenstein and M. Rose. Safe Tcl. Available at `ftp://ftp.fv.com/pub/code/other/safe-tcl.tar.Z`, 1995.

[CMM97] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: A multiagent system for distributed resource allocation. In *Proceedings of the International Conference on Autonomous Agents*, pages 323–331, Marina Del Rey, CA, February 1997.

[Gra96] Robert S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proceedings of the 1996 Tcl/Tk Workshop*, pages 9–23, July 1996.

[Gra97] Robert Gray. *Agent Tcl: A flexible and secure mobile-agent system.* PhD thesis, Dept. of Computer Science, Dartmouth College, June 1997.

[GSW97] Alok Gupta, Dale O. Stahl, and Andrew B. Whinston. Priority pricing of integrated services networks. In Lee McKnight and J. P. Bailey, editors, *Internet Economics*. MIT Press, 1997. To appear.

[KGN+97] David Kotz, Robert Gray, Saurab Nog, Daniela Rus, Sumit Chawla, and George Cybenko. Agent Tcl: A flexible mobile-agent system. *IEEE Internet Computing*, 1(4):58–67, July/August 1997.

[RGK] Daniela Rus, Robert Gray, and David Kotz. Transportable information agents. *Journal of Intelligent Information Systems*. To appear.

[SL95] Tuomas W. Sandholm and Victor R. Lesser. Equilibrium analysis of the possibilities of unenforced exchange in multiagent systems. In *Proceedings of the Fourteenth Interna-*

tional Joint Conference on Artificial Intelligence (IJCAI-95), pages 694–701, Montreal, Canada, 1995.

[SL96] Tuomas W. Sandholm and Victor R. Lesser. Advantages of a leveled commitment contracting protocol. In *Thirteenth National Conference on Artificial Intelligence*, pages 126–133, Portland, Oregon, 1996.

[Tan95] Mike Tanner. *Practical Queueing Analysis.* McGraw-Hill Book Company, London, 1995.

[Vic61] William Vickery. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[Wel96] Michael P. Wellman. Market-oriented programming: Some early lessons. In Scott H. Clearwater, editor, *Market-Based Control*, chapter 4, pages 74–96. World Scientific, Singapore, 1996.

[WHH+92] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[Whi94] James E. White. Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, 1994.

[WW96] Carl A. Waldspurger and William E. Weihl. An object-oriented framework for modular resource management. In *Proceedings of the Fifth International Workshop on Object Orientation in Operating Systems*, pages 138–143. IEEE Computer Society Press, October 1996.