Mobile-Agent Planning in a Market-Oriented Environment

Dartmouth College Department of Computer Science

Technical Report PCS-TR99-345, Revision 1

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

Original May 3, 1999
Revision May 20, 1999

Abstract

We propose a method for increasing incentives for sites to host arbitrary mobile agents in which mobile agents purchase their computing needs from host sites. We present a scalable market-based CPU allocation policy and an on-line algorithm that plans a mobile agent's expenditure over a multihop ordered itinerary. The algorithm chooses a set of sites at which to execute and computational priorities at each site to minimize execution time while preserving a prespecified budget constraint. We present simulation results of our algorithm to show that our allocation policy and planning algorithm scale well as more agents are added to the system.

1 Introduction

A mobile-agent system provides an environment that allows user programs (mobile agents) to voluntarily relocate and resume execution at another host site. Mobility is especially useful in reducing network latency and in operating in disconnected environments [LO98]. These qualities make mobility an attractive option for isolated applications and closed administrative domains, but the application of the technique has much greater potential. An important issue is providing hosts incentive to offer service for arbitrary mobile agents.

There are several techniques to protect host sites from visiting agents [Moo98], but hosts will always suffer from higher loads induced by visiting agents. To mitigate this problem we investigate the possibility of agents compensating host sites by purchasing computational resources from hosts [Tsc97, BKR98]. Resources include, but are not limited to, processor time, storage, and network access, as well as abstract value-added services.

In this paper, the resource on which we focus is CPU priority which we use to approximate general computational priority among agents. We present a lottery-based CPU scheduling policy and planning algorithms to allow mobile agents to plan expenditure of multi-hop itineraries. We describe a simulation system for these algorithms. Our experiments show that our algorithms allow agents to complete lengthy trips with high confidence in environments with bursty resource contention. Additionally, we show that our lottery-based allocation policy scales well as the number of agents in the system increases.

The value of a mobile-agent system is dependent on both the number of host sites that an agent may migrate to as well as the number of other agents with which an agent may interact. Conventional wisdom is that the value of a network increases quadratically with the number of users and sites that have access to it. We believe this also holds true for a mobile-agent system.

Given the importance of the number of sites, we would like to encourage site owners to open up access to their resources to the entire community of mobile-agent users. The resources used by mobile agents are general and often difficult to analyze. Not only do hosts sacrifice access to their own resources, but there are security risks inherent to providing any additional network service.

We propose that hosts be compensated for these factors through agents using a scarce verifiable electronic currency to purchases all their computational and information needs. We see several benefits of mobile agents participating in markets to access their computational needs. Markets limit the extent of agents' impact, provide simple a means of agent coordination through prices, and facilitate flexible administrative domains.

Because currency is scarce, agents' activities are limited. The extent of denial-of-service attacks are limited and, assuming that prices are efficient, a host would actually benefit from such an "attack," though an offender would not be able to carry out such an action for long.

In market systems, there is generally a strong correlation between demand and higher prices. By providing agents with the price of services, they become aware of their environmental impact. We would like to use this effect to give mobile agents incentive to avoid congesting a site. A possible solution for agents crowding a site is to either wait until a more appropriate time to execute or choose a less congested site.

2 An Allocation Policy

To encourage agents to act rationally, a reasonable price mechanism must be in use. In this section we explore an instance of a pricing mechanism where owners of priority-based resources (CPU's) lease shares, or tickets, for access. The quality of service a ticket holder receives is proportional to the number of tickets held relative to the number of tickets in circulation, creating a modified lottery or round-robin scheduling system.

The resource owner can fix the price of leasing tickets and let the number of tickets in circulation float; the real price of computation is determined by the number of tickets held by agents. Two major benefits of this pricing scheme are that the computational responsibilities of the price mechanism are distributed among the users and it is easy to implement.

Many traditional priority-based computational resources are allocated using priority queues or priority stratified round-robin schedulers. These systems rely on users having heterogeneous preferences to stratify priorities. Such schemes do not scale well as user preferences become more diverse, however. After the third or fourth priority level, no real service is left to allocate among users [GSW97, BKR98].

Our allocation policy asks each agent to submit a linear function describing the number of tickets each agent would buy given the number of shares held by competing agents. We restrict the function to be linear and decreasing.[1] Each time an agent arrives or departs the server, the server searches for a ticket circulation where each agent is content with the value and size of its share, a Nash equilibrium, and reallocates the new circulation according to agents' bidding functions.

Because agents cannot purchase negative number of shares, the bid functions are not convex. Thus a Nash equilibrium is not necessarily guaranteed to exist. We find that in practice, using a bisection search, we can find an ϵ-Nash equilibrium, where participants wish to change their bids by only a small amount.

3 The Model

The model that we examine is one where each agent is given a fixed endowment of currency and an ordered set of tasks to complete. Each task may be executed at one of several sites of varying capacity. Agents must choose a set of sites to visit and the quality of service desired at each site. The goal is to minimize time of execution while preserving budget constraints.

3.1 Lottery Allocation

At each site they visit, agents decide to rent some number of tickets at a fixed price per second. Tickets represent relative computing priority. The agent can then consume a computing share proportional to its ownership of lottery tickets relative to the total number in circulation.

[1] Section 3.2.1 justifies restricting bid functions in this way.

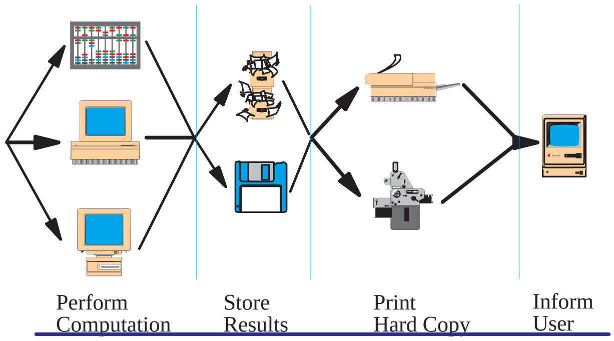

Figure 1: An example of an agent's itinerary. At each hop of the schedule, there may be multiple host sites willing to accept the agent's task.

Using this style of resource allocation, it is possible to describe the load of a host system with just a few parameters: ticket circulation and a performance metric of the host. Visiting agents do not need to disclose any information to participate in the market, other than the quantity of tickets desired to be purchased. The accuracy of a host's published parameters can be tested through benchmarking an agent's performance.

The strategy of how an agent would participate in a computational lottery market resembles the Prisoner's Dilemma [Axe84]. Agents can buy large quantities of tickets, the analog of defection or confessing to the police. Alternatively they may attempt to receive better performance by all buying small amounts of tickets, which is the analog of cooperating.

While agents may collaborate to form a cartel by agreeing to buy no more than a given amount of tickets (one each) from a server, it is frequently to an agent's advantage to buy an extra amount of tickets to increase performance at the collaborators' expense. Axelrod [Axe84] finds agents are less likely to cooperate if they are anonymous or expect little future interaction, so our lottery scheduling works better as the number of participating agents increases. In addition, the host system has incentive to protect the identity of visiting clients.

3.2 Utility Maximization

We need a metric for quality of service on which to derive users' utility. Frequently, quality of service is indicated by end-to-end latency of the task or by rate of computation. The accuracy or precision of the process is important in some cases, for example in a database query. The work in this paper only considers utility to be a function of the rate at which an agent computes. We assume that an agent has K tasks to complete and each task may be completed at one of any number of comparable host sites.

We wish to maximize the expected performance of all tasks in an agent's itinerary. c_k, M_k, and n_k represent the capacity in instructions per second, ticket circulation, and the agent's ticket holdings at the site where the kth task is executed. $Q = \sum_{k=1}^{K} q_k$, is the total quantity of computation in instructions to be done. We now compute an agent's expected rate of computation in Equation 1. The numerator is the sum of all job sizes and the denominator is the expected completion time, the quotient of the kth job size and expected rate of computation. The rate of computation is just the agent's ticket share,

n_k, relative to the total circulation, $n_k + M_k$, weighted by the processing capacity of the host, c_k.

$$U = \frac{Q}{\sum_{k=1}^{K} \mathrm{E}\left[\frac{q_k(n_k+M_k)}{n_k c_k}\right]} \tag{1}$$

We must also formalize the budget constraint. I represents the agent's initial budget endowment constraining the the sum of the amounts spent at each site, $p_k n_k t_k$, where $t_k = q_k(n_k+M_k)/c_k n_k$ denotes the time spent at the kth site, the quantity of the job size divided by the rate of computation. p_k is the price of renting a ticket at the kth host for a fixed time unit. Substituting the value of t_k into the product $p_k n_k t_k$ and taking the expected value, we derive the budget constraint to be:

$$I - \sum_{k=1}^{K} \frac{p_k q_k (n_k + \mathrm{E}[M_k])}{c_k} \geq 0 \tag{2}$$

To solve the problem of how many shares an agent should buy, we initially make some assumptions:

1. Once an agent chooses a site, it may not choose another until its current task is completed.

2. Each portion of the itinerary has only one site alternative.

3. The size of each agent's task is small compared to the sum of the sizes of the other tasks currently executing at any host site.

4. The network of hosts is at or near equilibrium.

5. The capacity, ticket price, and current ticket circulations of all hosts are available to agents.

We will later relax Assumptions 2, 3, and 4.

When Assumption 2 is dropped, the problem of choosing the optimal expected path is NP-complete if the agent has the estimated costs and execution times at each host. We note that the problem is the constrained shortest path problem [AMO93] and show a transformation presented by Ahuja *et al* [AMO93] from the knapsack problem [GJ79] to our problem. The constrained shortest path problem (CSP) is given a directed graph whose edges have associated costs and lengths, a path length, l, and a cost constraint, c, to find a path of at connecting two points where the sum of the path's edge cost is at most a c and the sum of the path's edge lengths is at most l.

The problem is in NP; a solution can be verified in linear time by summing the costs and lengths of all edges in the solution. We can express any knapsack problem in terms of CSP by creating a lattice graph in Figure 2. The graph is divided into three levels. The nodes in the top level, labeled i', represent the choice of placing the ith object in the knapsack. The nodes in the bottom level, labeled i'', represent the choice of excluding the ith object. The edge from i to i' has cost equal to the weight or volume, w_i, of the ith object, and distance equal to the negative value, v_i, of the ith object. All other edges have zero cost and distance.

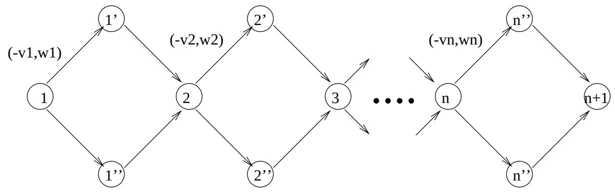

Figure 2: A reduction of the knapsack problem to the constrained shortest path problem.

Since any knapsack problem can be expressed as a CSP problem and CSP is in NP, CSP is NP-complete. We use Assumption 2 to create an estimator for the general case in Section 3.2.2.

Assumption 3 allows us to reason about the actions of a single agent without attempting to second guess the rest of the market. Agents construct a purchasing policy, or bidding function, based upon the assumption. The bidding functions are then submitted to the service providers who attempt to find a resource allocation that satisfies every agent's bid function.

Our next task is to derive agents' purchasing policy that will minimize agents' execution time. A purchasing policy must account for estimating the random variables, M_k's, in Equations 2 and 1, the topic of the next subsection.

3.2.1 A Simple Estimator

Efficient expenditure planning must forecast future network conditions. In our problem, agents are concerned with the congestion of resources represented by the size of hosts' ticket circulations. We now describe a method for agents to estimate future resource contention based upon current conditions.

Assumption 4 implies that the load of every host in the network is approximately equal to the current host's load relative to price and capacity. The agent observes that the first site's ticket circulation is m_1. We set $M_1 = m_1$ and weight hosts' loads according to capacity and price yielding:

$$\mathrm{E}[M_k] = \frac{(p_1/c_1)}{(p_k/c_k)} m_1 \tag{3}$$

The use of the estimator in Equation 3 makes sense if there is a reasonable flow of agents requesting service. Incoming agents will choose to hold ticket leases with the highest value. Once a site becomes crowded, the value of each additional ticket sold drops, new agents will choose to execute at sites with higher ticket values, and the local population growth will slow.

Using the Lagrange function [Lan87], Equation 4, we minimize the denominator of Equation 1, $t(n_1, \ldots n_K, m_1)$, under the budget constraint, Equation 2.

$$\begin{aligned} L(n_1, \ldots n_K, m_1) &= t(n_1, \ldots n_K, m_1) \\ &\quad -\lambda(I - e(n_1, \ldots n_K, m_1)) \end{aligned} \tag{4}$$

Where $t(n_1, \ldots n_K, m_1)$, the denominator of Equation 1, and $e(n_1, \ldots n_K, m_1)$, the budget constraint from Equation 2, denote the expected amount of time an agent takes to perform its itinerary and the expected expenditure completing it, respectively, given the ticket share purchases at each site and the ticket circulation at the first host site. The resulting solution gives a ratio for the share of computation at any two hosts:

$$\frac{n_k}{n_i} = \sqrt{\frac{p_i/c_i}{p_k/c_k}} \qquad (5)$$

Using Equation 5 with the budget constraint, we obtain the optimal share to purchase at the first site:

$$n_1 = \frac{I - \frac{m_1 p_1}{c_1} \sum_{k=1}^{K} q_k}{\sum_{k=1}^{K} q_k \sqrt{\frac{p_1 p_k}{c_1 c_k}}} \qquad (6)$$

If the agent assumes that the current load of future sites is equal to the future load, however, the optimal number of tickets to buy at the first site is:

$$n_1 = \frac{I - \sum_{k=1}^{K} \frac{p_k q_k m_k}{c_k}}{\sum_{k=1}^{K} q_k \sqrt{\frac{p_1 p_k}{c_1 c_k}}} \qquad (7)$$

Assumptions 3 and 4 imply that a newly arriving agent's request for computing lottery tickets will not effect the competition very much. That is, agents already present will not significantly change their ticket holdings in response to one more agent arriving.

In order for a host to obtain a solution in which all agents are content with their ticket holdings, a host must repeatedly query agents for their purchases given the result of the last iteration. Each iteration drives ticket holdings towards a Nash equilibrium. Note that Equation 6 is linear with respect to m_1. On this note, we would like to analytically compute a fixed point solution where all agents present at the host site will hold a stable ticket share given their competitors' holdings. Let there be L agents at the site and each lth agent submits parameters

$$a_l = -\frac{\frac{p_1}{c_1} \sum_{k=1}^{K} q_k}{\sum_{k=1}^{K} q_k \sqrt{\frac{p_1 p_k}{c_1 c_k}}} \qquad (8)$$

and

$$b_l = \frac{I}{\sum_{k=1}^{K} q_k \sqrt{\frac{p_1 p_k}{c_1 c_k}}} \qquad (9)$$

to describe each client's linear bidding function. a_l and b_l are the coefficients of n_1 in Equation 6. We omit subscripting of all the parameters on the right-hand side of the Equations 8 and 9 with l to avoid clutter.

We can calculate the number of shares that clients buy as an iterated process. At the ith iteration, the lth client buys a number of shares specified by the bidding function parameters and the number of competing shares at the previous iteration, $m_{i-1} - n_{l_{i-1}}$ to yield:

$$n_{l_i} = \max(0, a_l(m_{i-1} - n_{l_{i-1}}) + b_l) \qquad (10)$$

The total number of tickets sold at the ith iteration is then:

$$m_i = \sum_{l=1}^{L} \max(0, a_l(m_{i-1} - n_{l_{i-1}}) + b_l) \qquad (11)$$

The max() function restricts agents to buying a non-negative number of shares.

We cannot analytically solve Equations 11 and 10, but if we ignore the fact that the $a_l(m_{i-1} - n_{l_{i-1}})$ terms may be negative, set each $n_{l_i} = n_{l_{i-1}}$ and $m_i = m_{i-1}$, and solve, we get a good start on the search process for a ticket share equilibrium. Every time an agent arrives or leaves a site, the server runs a binary search shown in Figure 3.2.1 to set the circulation. The search process runs until the ticket circulation oscillates within a set tolerance, ϵ, or exceeds a set number of iterations, *limit*.

```
ServerAllocate
m ← Solving fixed point of
      Equations 11 and 10
lastm ← m
i ← 0
do
    lastm ← (m + lastm)/2
    m ← 0
    foreach client l do
            m += a_l * lastm + b_l
    i++
while ((|m − lastm| > ε or i < 2)
    and (i < limit or m < 0))
```

Figure 3: The processor allocation algorithm used by servers.

We assume that host sites are able to augment q_1 for each agent. It may be the case that the series, $m_1, m_2, m_3, \ldots$, does not converge to a positive value. Experimentally, though, we find that the series converges more than 99% of the time, most of the time in fewer than ten iterations. When the series does not converge within *limit* iterations, we choose the last positive ticket circulation encountered in the search.

3.2.2 Considering Site Alternatives

We would like to expand our planning mechanism to include information about more than one host site per task. To handle site choice, we augment Equation 6 slightly. We assume that agents can observe ticket circulations at prospective hosts for the next task to be completed. For later tasks, we use the means of the prices and capacities of sites in each service group.

Agents then choose host sites by constructing an estimate of expected completion time for itineraries beginining at each prospective host using the averaged prices and capacities for prospective hosts of the second and later tasks. Again, agents assume that the reaction of other agents already at the prospective hosts will be neglibible. The host site with the lowest estimated completion time is then chosen.

For example, an agent may have two tasks. There are two sites able to accept the first task and two more sites able to accept the second. The second pair of sites have capacities 2.0 and 2.5, ticket circulations 5.2 and 7.0, and ticket prices 1.0 and 1.5, respectively. The agent uses Equation 6 twice, setting $c_2 = 2.25$ and $p_2 = 1.25$, to determine the rate of compuation if the agent were to begin its itinerary at either of the first pair of sites. In the calculations, m_1 is set to the value of the current ticket circulations of each of the first pair of hosts. The agent then chooses the faster of the first pair of hosts.

We simulate strategies using both the future site averaging and the reaction estimate assumptions in the next section.

4 Simulation

We wish to see how our strategies perform in planning resource expenditure. To do so, we simulate a network with a fixed number of servers using the DaSSF simulation package [NL99]. The servers are partitioned by the service that they provide. Our simulations represent a network of eight services each provided by eight hosts resulting in a network with 64 host sites. Host site capacity is normally distributed to make site choice more meaningful.

Agents in the system are created at a Poisson arrival rate with exponentially sized jobs (q_k) comprising itineraries whose length (K) are exponentially distributed. Endowments (I) are the product of a normally distributed random variable and the agents' job sizes ($Q = \sum q_k$).

Agents choose a site based upon the congestion of the network, represented by site ticket circulation. Once the agent chooses the site, it commits to attempting to complete the current task at the chosen site.

We now present several strategies that agents can use to choose sites and bid. Several are simple baseline comparisons.

4.1 Cheapest Available

Our first comparison algorithm, CHEAP shown in Figure 4.1, chooses the site which will complete the next task with the lowest estimated expenditure. The predicate `cheaperTask` returns true if the site represented by the first argument requires lower expenditures than the site represented by the second argument for executing the task represented by the third argument. ϵ is the smallest number of shares that hosts will lease to an agent. Once the agent chooses the site, the agent bids for the smallest amount of shares possible until the job completes.

```
CHEAP
foreach task i
    q_max ← ∅
    foreach possible site j
        if cheaperTask (q_max, j, i)
            q_max ← j
jumpTo q_max leasing ϵ shares
```

Figure 4: The CHEAP planning algorithm.

With little competition, budget expenditure is a weak constraint and one would expect this algorithm to complete jobs with a high degree of certainty.

4.2 Random

As another comparison, we have implemented a strategy we call RAND which is shown in Figure 4.2. Each site offering a given service has an equal chance of being chosen by the algorithm. Once a site has been chosen, the algorithm uniformly chooses between zero and 90 percent, non-inclusive, of the budget to spend at the current site.

4.3 Equilibrium Assumption

We implement a strategy from Equation 6 to choose sites and bid for shares. We find that

RAND

foreach task i

  $host \leftarrow$ `uniform` {hosts offering i}

  `jumpTo` $host$

    `spending` (`uniform` $(0, 0.9I)$)

Figure 5: The RAND planning algorithm.

frequently agents using this strategy bid too aggressively. That is, agents buy too many shares early in their itineraries to allow reasonably fast completion of later tasks. To account for the strategy's myopia, we multiply agents' job size estimates by a constant factor for purposes of bidding. Figure 6 shows the results of our empirical search for a good bias factor. In the experiment, all agents use the same job size bias factor and agents arrive at a rate so that system utilization is half the system capacity. Empirically, we find that 2.2 is a good bias factor.

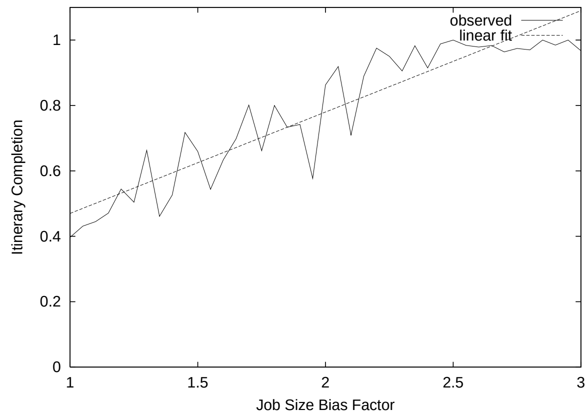

Figure 6: Success rate of agents using the EQ algorithm versus the job size bias factor.

EQ

foreach task $q \in Q$

  $q \leftarrow q * bias$

foreach task i

  $q_{max} \leftarrow \emptyset$

  $t_{min} \leftarrow \infty$

  foreach possible site j

    $t \leftarrow$ `tripTime`

      `buying` Eqn 6 shares

    if $t < t_{min}$

      $q_{max} \leftarrow j$

  `jumpTo` q_{max} `withBidFunc` Eqn 6

Figure 7: The EQ planning algorithm.

4.4 Complete Knowledge

The next strategy, COMP, is exactly like the previous, except that the current load of all servers to be used later in the itinerary is known at the time agents choose a site and bid for tickets. As with prices and capacities, measures of load for future sites is averaged over all sites offering the same service. This strategy is less sensitive than the EQ strategy to fluctuating loads, so we do not bias job size. The bidding function for each agent is Equation 7.

We augment the EQ algorithm with the bidding function in Equation 7 to yield the algorithm shown in Figure 4.4, the COMP planning algorithm.

4.5 Simulation Results

We investigate two general classes of environments: one where all participating agents

```
COMP
foreach task i
    q_max ← ∅
    t_min ← ∞
    foreach possible site j
        t ← tripTime
            buying Eqn 7 shares
        if t < t_min
            q_max ← j
    jumpTo q_max withBidFunc Eqn 7
```

Figure 8: The COMP planning algorithm.

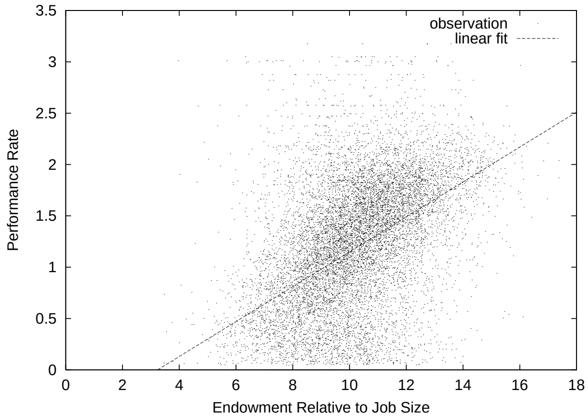

Figure 9: Agent endowment relative to job size versus performance with χ^2 fit.

use the same bidding strategy, and one where each agent uses a randomly chosen strategy.

We would first like to verify that agents that pay more receive better service than agents who pay less. Figure 9 shows observations of the amount that agents using the EQ strategy spend relative to the size of their jobs versus the performance that they receive. We also plot a χ^2 linear fit of the data. The graph confirms that agents' expected performance is dependent to expenditure.

Figure 10 shows the rate at which agents complete their itineraries compared to the job arrival intensity rate. The rate of 6.4 jobs per time unit is the system's maximum capacity. Agents using the EQ strategy complete their itineraries over 97 percent of the time until system utilization exceeds 85 percent of capacity. COMP performs marginally better.

Agents using the CHEAP strategy always complete their task. We omit plotting the outcome. The strategy's success is due to the fact that budgets dwarf expenditure. Agents

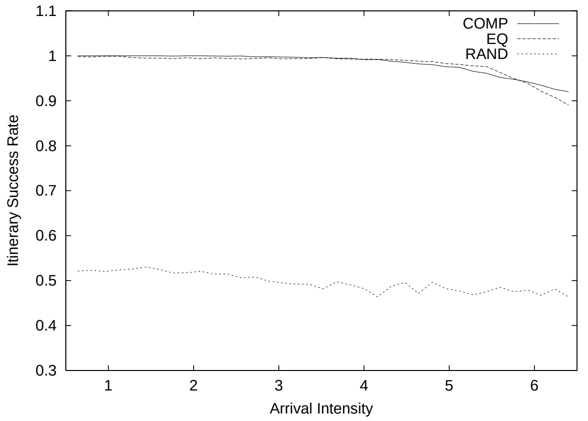

Figure 10: Itinerary success rate when all agents use a single strategy.

using the CHEAP strategy cooperate to keep the effective price of computation to negligible levels and effectively eliminate budget constraints. All agents receive equal computational priority, however, and there is little reason to enforce a market system if all participating agents use the CHEAP strategy.

Figure 11 shows the mean performance

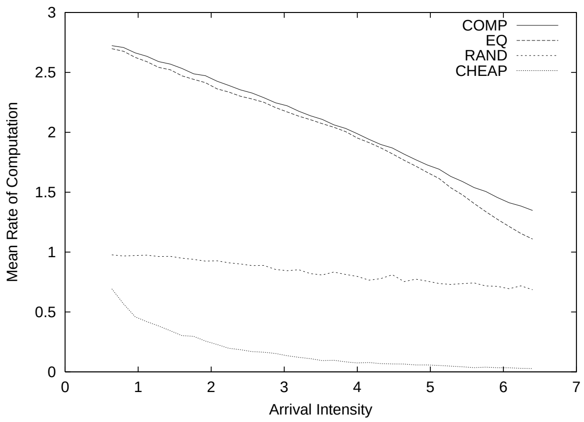

Figure 11: Average computational rate for completed itineraries when all agents use a single strategy.

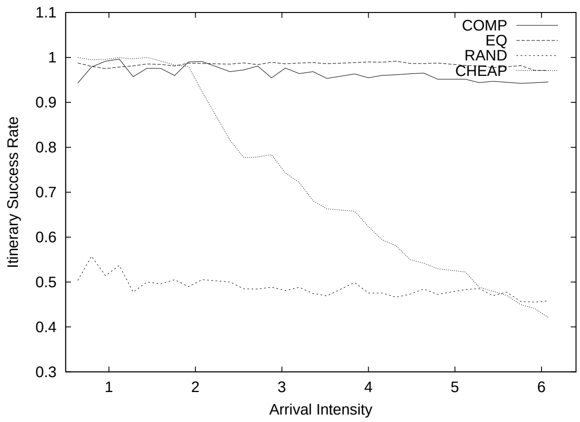

Figure 12: Success rates of a agents in a scenario where each strategy is used by one quarter of the population.

rates of agents using various strategies, again where all agents in the scenario have the identical strategies. The mean host site capacity is 2.0, but in light loads, agents are able to find access to faster hosts and raise the mean rate of computation above the mean host capacity.

Again, the EQ and COMP strategies have similar results. The RAND strategy does not perform nearly as well as EQ and COMP since choosing host sites randomly tends to result in pockets of moderate congestion. Additionally, the wide spread in bids leads to performance disparities that lower the mean computation rate.

Agents using the CHEAP strategy have dismal performance; they always pick the cheapest service provider regardless of the performance. The result is that a few providers handle much of the load and the more expensive ones are left empty.

Finally, we are interested in how the different strategies compete against each other. In our next simulation, when an agent is created, it is assigned one of the four strategies uniformly.

Figure 12 shows the itinerary completion success of the four strategies competing in a common environment. The strategies, with the exception of CHEAP, perform similarly as they do in homogeneous environments. The CHEAP strategy's success is affected by the other strategies crowding the agents using CHEAP. Once the ticket circulation rises above a certain level, it becomes impossible for a CHEAP agent to complete its task.

Figure 13 shows the mean rate that agents compute their tasks in job units per time unit. Again, the results are similar to those in the homogeneous environments.

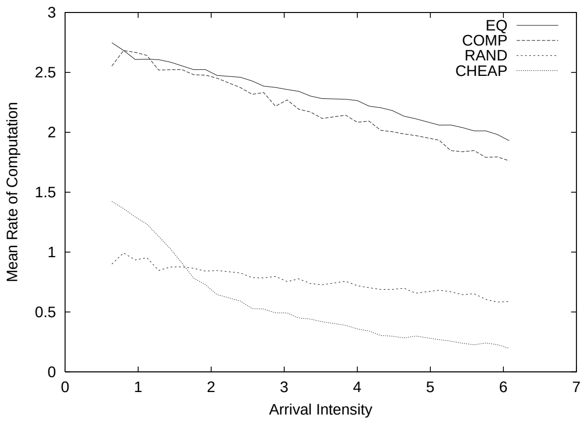

Figure 13: Average rate of computation for completing agents in an environment where each strategy is used by one quarter of the population.

5 Discussion

Both EQ and COMP strategies perform well. Good performance is evident in Figures 11 and 13. While the network load is under half capacity, the majority of agents are able to achieve service better than the average host can provide.

Since the level of service degrades gradually with increasing arrival intensities, we believe that our lottery allocation policy scales nicely and that the EQ and COMP strategies will work well in many environments.

In our experiments, complete knowledge of the network state does not give agents significant gains in performance. In part, this is because loads of servers across services are correlated. If the loads were uncorrelated (or perhaps even negatively correlated), the equilibrium assumption used in the EQ strategy would hinder agents' performance, while agents using complete knowledge would be less effected.

The equilibrium assumption, however, does have one benefit when shares are updated: it is not necessary to obtain the loads of all servers in the network. This feature is especially important if the network is disconnected as in the case of many mobile-agent applications.

Another benefit of using the EQ and COMP strategies is their simplicity. Much of the EQ and COMP algorithms' simplicity is is due to the fact that the agents do not attempt to preserve savings; they attempt to spend all of their funds on computation. This behavior justifies the use of a linear bid function.

If savings were an issue, however, linear functions would not suffice. Agents would then desire to bid small amounts in situations with little congestion as well when resource contention is high. Agents would bid most under moderate resource contention. Dispensing agents' need to save facilitates equilibrium finding and avoids the problem of measuring the value of savings versus faster execution.

5.1 Structure

In running many simulations and devising bidding algorithms we have noticed a few properties of our lottery-based resource allocation system. Information regarding the distribution of ticket circulation at host sites certainly effects planning decisions on the part of agents. We observe that the distribution of ticket sales at sites is roughly Gaussian. Figure 14 shows histograms of ticket frequencies at two different sites with 50 percent load. Each histogram is generated by collecting the ticket circulation whenever an agent arrives or departs a site.

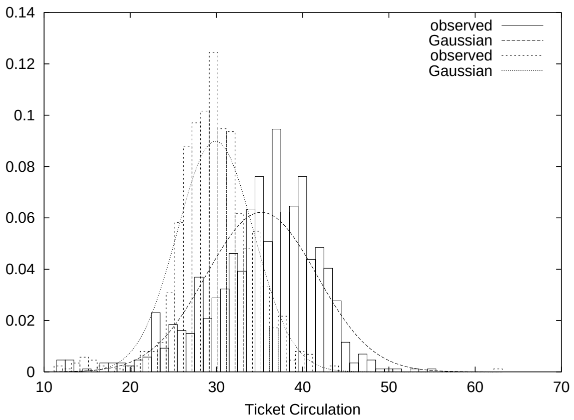

Figure 14: Histogram of ticket circulation at two sites over a run with a relative load of 50% and Gaussian distributions with the corresponding mean and variances.

We believe that this distribution is heavily dependent on the distribution of the sum of agents' endowments relative to their job sizes (also Gaussian). We believe that the distribution of wealth at sites is valuable knowledge for expenditure planning and our future expenditure planning algorithms will use the distribution to estimate resource congestion.

The distribution of ticket sales at sites will become important if we wish to consider the possibility of allowing agents to restart or continue their unfinished current execution at another site. Agents could estimate their expected time savings given the time taken to jump across the network (a feature we do not yet model) and the mean and variances of other sites' tickets sales.

We now turn our attention to general properties of our system at equilibrium. Specifically, we are interested in the relationship between price of tickets and congestion, the ticket circulation relative to system capacity.

At equilibrium, the relationship of price to congestion is strictly decreasing. The argument is simple: agents will choose cheaper service from faster hosts over more expensive service from slower hosts.

Currently, we do not model agents' ability to split the execution of one task across multiple servers. Breaking up computation is not unreasonable if the mobile-agent system supports a transparent jump command, like D'Agents' `agent_jump` command. An agent can execute a portion of its task at one server and then relocate to another host to continue the rest of the task. The resulting performance is a weighted average of the two sites with an amount of overhead to account for relocating the agent.

If we allow splitting of tasks, then we may make further assumptions on the relationship between price and congestion; at equilibrium the relationship may not be concave downward. Breaking up the execution allows agents to use portfolio strategy to blend performance of multiple sites to achieve a weighted average of site performance. Hence if the relationship becomes concave downward, agents will prefer splitting their execution between sites cheap congested sites and expensive fast sites rather than jump to sites less extremely characterized sites. The argument is illustrated in Figure 15.

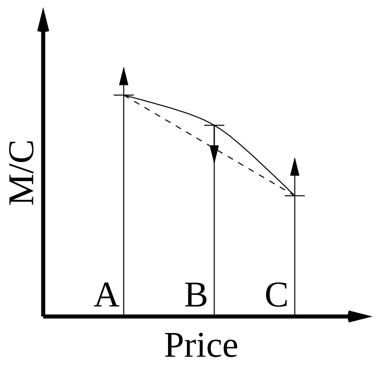

Figure 15: Relative congestion as a function of ticket price at equilibrium may not be concave downward if agents' jobs can be broken up to be executed at multiple sites. If the function is concave downward as represented here, agents will choose to execute parts of their jobs at A and C until the weighted performance of such a portfolio becomes at least as congested as the performance at site B.

One final note on price concavity: if agents use mean metrics for planning expenditure as done in Section 3.2.2 and the price-congestion relationship is concave up, the price-congestion pair representing the average of several hosts will provide a conservative estimator.

6 Related Work

Waldspurger [WW94] uses lotteries as a resource allocation model to manage computational resources such as network bandwidth, processor time, and memory. Two major differences between standard lotteries and ours is that tickets are leased, not bought, and are consumed only when the lease expires. Scheduling can be done in a simple round-robin time slicing manner.

An implementation a lottery market-based system is in the Geneva Messengers [Tsc97] mobile-agent system where a currency system is used to allocate CPU time as well as memory for mobile agents. Processing resources are allocated through agents buying tickets. At each quantum, a ticket is chosen and the owning agent is given access to the CPU for the quantum. One feature of lotteries implemented in this way, is that the while the price of computational priority may fluctuate, the price of a quantum of CPU time is fixed, facilitating budget planning. Pricing in this manner does not enforce a positive correlation between the demand and the price of a scarce resource, however, giving agents little incentive to balance network load.

Memory in the Messengers system is also allocated using a currency system. Agents "sponsor" persistent blocks of memory. A block of data is endowed with currency by interested parties (readers). Periodically, the sponsored blocks' accounts are charged and blocks having depleted accounts are flushed from the system. The rate at which blocks are charged varies over time depending on the contention for additional block space.

While Messengers has market resource allocation mechanisms, the system presents no means for expenditure planning on the part of agents.

Moizumi [Moi98] attacks similar mobile-agent problems. He derives a general procedure for mobile agents to plan an itinerary visiting a known set of hosts using dynamic programming is presented. Much of the work is based on NP-completeness work through reduction to the Traveling Salesman Problem and approximation solutions. Our work takes a different slant; we assume that the basic sequence of computation is predetermined and that refinements of location and priority decisions are made along agents' travels.

In our investigations concerning the applicability of our system to load balancing, it is certainly worth mentioning that it has been shown that heterogeneity of preferences can improve load balancing substantially by encouraging adaptive systems to further explore the parameter space than agents in a homogeneous populations [SST95].

7 Summary

We believe that mobility would be of far greater use to software developers if a larger number of host sites would be willing to host arbitrary agents. There is little reason for sites to accept foreign mobile agents, however. To remedy this shortfall, we propose that agents purchase the resources that they consume from hosts.

We present a market-based CPU priority-allocation policy and simple algorithms that allow mobile agents to plan multi-hop trips through a network of host systems providing various services. The algorithms that we present produce both a route through the network and the desired computational priority to be received at each site on the route. The constructed plans minimize agent execution time while preserving a fixed budget constraint.

Our simulations show that the algorithms scale smoothly. Evidence of our algorithms' effectiveness is that we are frequently able to plan routes through a network with performance greater than the average host is capable of providing and that we are able to complete agent itineraries with empirically high confidence.

Acknowledgments

This paper describes research done in the Mobile Agents Laboratory[2] at Dartmouth. This work is supported in part by the Navy and Air Force under contracts ONR N00014-95-1-1204 and MURI F49620-97-1-0382, Rome Labs under contract F30602-98-C-0006, and DARPA under contract F30602-98-2-0107. We would like to thank Jason Liu for his invaluable help using DaSSF.

[2]http://agent.cs.dartmouth.edu

References

[AMO93] Ravinda K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Upper Saddle River, NJ, 1993.

[Axe84] Robert M. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, NY, 1984.

[BKR98] Jonathan Bredin, David Kotz, and Daniela Rus. Market-based resource control for mobile agents. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 197–204, Minneapolis, MN, May 1998. ACM Press.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, NY, 1979.

[GSW97] Alok Gupta, Dale O. Stahl, and Andrew B. Whinston. A stochastic equilibrium model of Internet pricing. *Journal of Economic Dynamics and Control*, 21:697–672, 1997.

[Lan87] Kelvin Lancaster. *Mathematical Economics.* Dover Publications, Mineola, NY, 1987.

[LO98] Danny Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley, Reading, MA, 1998.

[Moi98] Katsuhiro Moizumi. *The mobile-agent planning problem.* PhD thesis, Thayer School of Engineering, Dartmouth College, 1998.

[Moo98] Jonathan T. Moore. Mobile code security techniques. Technical report, University of Pennsylvania, 1998.

[NL99] David Nicol and Xiaowen Liu. DaSSF: the Dartmouth scalable simulation framework, 1999.

[SST95] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

[Tsc97] Christian F. Tschudin. Open resource allocation for mobile code. In *Proceedings of The First Workshop on Mobile Agents*, pages 186–197, Berlin, April 1997.

[WW94] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *First Symposium on Operating System Design and Implementation (OSDI)*, pages 1–11. USENIX Association, 1994.