MARKET-BASED CONTROL OF MOBILE-AGENT SYSTEMS

A Dissertation
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of
Doctor of Philosophy
in
Computer Science
by
Jonathan L. Bredin
DARTMOUTH COLLEGE
Hanover, New Hampshire
June 1, 2001

Examining Committee:

David Kotz (chair)

Daniela Rus

George Cybenko

Michael Wellman

Roger Sloboda
Dean of Graduate Studies

Abstract

Modern distributed systems scatter sensors, storage, and computation throughout the environment. Ideally these devices communicate and share resources, but there is seldom motivation for a device's owner to yield control to another user. We establish markets for computational resources to motivate principals to share resources with arbitrary users, to enforce priority in distributed systems, to provide flexible and rational limitations on the potential of an application, and to provide a lightweight structure to balance the workload over time and between devices. As proof of concept, we implement a structure software agents can use to discover and negotiate access to networked resources. The structure separates discovery, authentication, and consumption enforcement as separate orthogonal issues to give system designers flexibility.

Mobile agents represent informational and computational flow. We develop mechanisms that distributively allocate computation among mobile agents in two settings. The first models a situation where users collectively own networked computing resources and require priority enforcement. We extend the allocation mechanism to allow resource reservation to mitigate utility volatility. The second, more general model relaxes the ownership assumption. We apply our computational market to an open setting where a principal's chief concern is revenue maximization.

Our simulations compare the performance of market-based allocation policies to traditional policies and relate the cost of ownership and consumption separation. We observe that our markets effectively prioritize applications' performance, can operate under uncertainty and network delay, provide metrics to balance network load, and allow measurement of market-participation risk versus reservation-based computation.

In addition to allocation problems, we investigate resource selection to optimize execution time. The problem is NP-complete if the costs and latencies are constant. Both metrics' dependence on the chosen set complicates matters. We study how a greedy approach, a novel heuristic, and a shortest-constrained-path strategy perform in mobile-agent applications.

Market-based computational-resource allocation fertilizes applications where previously there was a dearth of motive for or means of cooperation. The rationale behind mobile-agent performance optimization is also useful for resource allocation in general distributed systems where an application has a sequence of dependent tasks or when data collection is expensive.

Acknowledgments

The D's supported my research for five years with enormous patience. Please excuse the omission of grant numbers here. Dave, your consistency and fastidious attention to detail are invaluable. Daniela, I owe any research acumen I have to you. Rajiv, you are a great collaborator. On the less obvious, but equally valuable contributions: Mom and John encouraged me during some dark moments. My father showed me the value of generosity. And Mina taught me the value of simplicity. Kier reminded me that it's always possible to smile. Arne cheerfully greeted me every morning at the lab.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As computing moves into the post-PC era [Hen99], computer-science research should investigate flexible efficient structures for resource allocation. The paradigm of ubiquitous computing places hordes of low-power computing devices in every corner of our environment. Efficient placement and use of these electronic devices usually involves resource sharing— computational and informational resources in particular— and realistic execution involves data compilation from several sources and, hence, resource consumption across the environment. If we can pare down resource requirements and allow devices to share or remotely access resources, we can enable computation in many more scenarios than previously possible.

Not only do devices become more economical when they share resources, there are many applications, from electronic commerce to entertainment, that can synergistically leverage multi-agent participation. A quality of these applications and many small devices is that they are portable across administrative domains. For example, an informational agent may gather weather forecasts from a database in at airport and later, at a hotel, access another database that features restaurant listings. The user can better choose a restaurant given the weather's impact on the aesthetic atmosphere or travel to the restaurant. The agent has to negotiate resource access from different principals in the scenario. Possibly, the agent may have to convince a

principal that there is a beneficial motive to access the resource.

We investigate the possibility that currency exchange motivates, or simply regulates, resource access. We establish a market where the agent exchanges with the principal scarce currency for resource access. We model the flow of data and computation through the network with the abstraction of the *mobile agent*— a user level process that can autonomously alter its execution location without any other alteration to its state. Our interests lie in the synthesis and regulation of environments that facilitate exchange of computation and data among mobile agents and their hosts. We promote the idea that a mobile agent purchases the computational resources it consumes from its hosts. We entertain the possibility that agents' and hosts' interests conflict, but we would like resource, information, and service exchange to occur in spite of the dissonant motivations.

There is prior computer-science research that leverages economics to distributively allocate resources. Many projects assume that decision-making agents have harmonic interests or are forced to participate in a market. Additionally, most existing work in computational-resource allocation attacks the problem of how to allocate a single-use good to an agent. We provide an environment where self-interested parties can competitively exchange computation, data, and electronic currency representing computing potential. We consider how an agent can tune its resource consumption across many sub-tasks to optimize its top-level performance. In concert with priority coordination, we explore how an agent may choose the resources it consumes.

We study two computational-resource allocation systems. In one, consumers collectively own the resources. We call this a closed-interest market because it restricts the motivations of resource owners and consumers. An open-interest market makes no such assumptions. We compare the incentives in the two markets and observe the cost of the separation of interests. The cost appears in the form barriers to information exchange, which inhibit agents and principals from disclosing their costs,

and uncertainty. We provide methods for an agent to negotiate priority under the uncertainty of information barriers as well as congestion volatility.

A problem that we attack in the dissertation is planning sequential resource consumption. We examine how an agent can budget its potential across a sequence of tasks when the performance and expenditure of one task affects other tasks in the sequence. Within the context of a shared-resource environment, we propose a mechanism that prioritizes agents by their currency holdings. We analytically optimize an agent's resource consumption to minimize end-to-end latency under its budget constraint and the allocation mechanism. The allocation forms a Nash equilibrium - a situation from which no agent can improve its performance through unilaterally changing its action.

We compare our allocation method with other ones to show the cost incurred in arbitrary agent prioritization (about 8%) and the performance benefits of our allocation over non-prioritized allocation (about 18%). Prioritization allows the system to complete a portion of its tasks when user requests dominate system capacity. Furthermore, our allocation and budgeting processes operate well under uncertainty in an agent's resource requirements and is no more sensitive to network latency than the other allocation methods we examine.

Under the allocation policy that we propose, we observe variance in agents' performance over time. We recognize that some applications may be able to tolerate the risk incurred from the volatility and others may not. To provide an environment in which agents with heterogeneous preferences for risk may operate, we explore the possibility that resource vendors issue reservations to risk-averse agents. We leverage financial option-pricing methods to evaluate reservations through prediction of resource contention. We show how agents and hosts exchange risk through mutually beneficial trade.

We attack resource allocation among agents whose interests conflict with their

hosts' interests and relax resource-ownership assumptions. We derive an algorithm that assesses the price of computation given that an agent has non-linear marginal value for computation. Through the use of this process, we show how an agent can derive a bidding strategy that maximizes its expected utility when the outcome of any number of actions within a sequence has associated uncertainty. Through the comparison of our simulations, we show the cost of separation of consumption and ownership interests.

To complete its tasks, an agent has to decide which resources to consume as well as the priority at which to consume. Complications arise from a circular relationship between resource selection and prioritization. An agent that optimizes its consumption of a selected set of resources adjusts the comparative and absolute cost differences between the resources. The context-dependent nature of resource quality feeds back into the selection problem.

We propose three methods that allow an agent to choose its hosts. We show that performance of a greedy algorithm compares favorably with more computational methods and we derive a separate algorithm that performs well when the network topology is unknown or latencies are volatile.

We wrap up the dissertation with a framework with which principals can use to describe resources and agents can use to discover resources, authenticate themselves, and negotiate access to the resources. The framework separates authentication, allocation, and consumption enforcement as orthogonal issues to allow administrators and software engineers flexibility in resource control. Our framework is a proof-of-concept to be used with existing service-discovery models, such as Jini [AOS+99], and electronic currency payment systems, such as Millicent [GMA+96] or Netcents [PHS98].

1.1 Road Map

We look at how an agent plans its resource consumption subject to a budget constraint and market participation. Section 1.2 sketches the general operation of computation markets. In Chapter 2, we fill in more details and further motivate our decision to study markets. In Chapter 3 we summarize work that leverage economics to solve problems in computer science, as well as relevant mobile-agent research and distributed resource-management packages. We describe an application model in Chapter 4 that we use to derive agent-planning and resource-allocation algorithms in Chapters 5 and 6.

We present allocation policies and resource-consumption planning algorithms for closed and open-interest markets in Chapters 5 and 6, respectively. Chapters 5 and 6 focus on resource allocation, and in Chapter 7 we look at algorithms that determine which hosts an agent should visit. Chapter 8 incorporates the notion of volatility in agents' performance and we propose mechanisms that allow an agent to sell its risk to its host. In Chapter 9 we present an application-program interface that allows agents to query the network for a resource, as well as reserve the resource. Finally, we relate our overall experiences using market-based control for computational-resource allocation in Chapter 10.

1.2 Computation Markets

Earlier in this chapter, we mentioned the idea of markets to allocate computation among mobile agents. Here we develop the idea further. We would like to motivate computational-resource owners to host arbitrary mobile agents. Mobile-agent accommodation exposes a host to denial-of-service attacks and consumes valuable resources.

Dwork and Nor [DN92, Mol01] propose a system for client-server interaction that

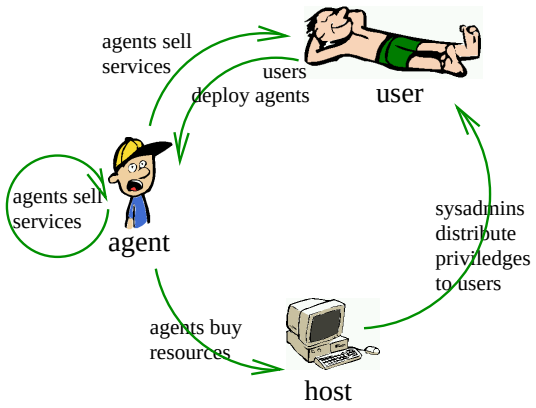

Figure 1.1: The flow of money in our computational market for mobile-agent resources.

indicates a service requester has a serious request through a proof that shows the requester expends as many computational resources as the service provider by computing hash-code collisions. In effect, the service provider is more likely to provide service if it sees that the client "bleeds" as much the server. The system prevents denial-of-service attacks, but is inefficient and does little to promote a principal to expose its resources to the general public.

We promote the notion that a mobile agent (a resource consumer) presents cryptographically signed messages, that represent electronic currency, to a host (a service provider) in exchange for computation and other resources. A user endows her agent with currency that the agent uses purchase computation from its hosts and sell services to other users and agents. Eventually, the currency accumulates at hosts, whose administrators redistribute the currency to users. Figure 1.1 shows how currency is distributed in our proposed computational markets. Figure 1.2 shows an example of the exchange of currency at a mobile-agent host site.

1.3 Mobile-Agent Applications

Before we proceed any further, it is worth mentioning two things. The resource allocation planning and techniques that we present in this dissertation are applicable

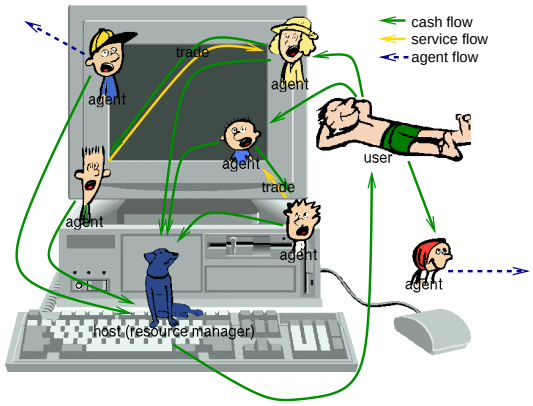

Figure 1.2: The exchange of currency between users, agents, and a host at a particular site.

Mobile agents

1. "reduce the network load...."
2. "overcome network latency...."
3. "encapsulate protocols...."
4. "execute asynchronously and autonomously...."
5. "adapt dynamically...."
6. "are naturally heterogenous...."
7. "are robust and fault-tolerant...."

Figure 1.3: Lange and Oshima's seven motivations for using mobile agents [LO99].

to many other scenarios, not just ones involving mobile agents. The second note is more involved: we justify why mobile-agent applications are important.

Lange and Oshima [LO98, LO99] cite seven reasons to motivate mobile agent use. We quote them in Table 1.3. Kotz and Gray [KG99] further motivate mobile agents in the context of dynamic network services for preference storage and proxies. They propose that mobile agents and mobile code will expedite server customization and serve as a proxy for user preferences, especially in disconnected and client-oriented environments.

We now back the claims that mobile-agent approaches benefit system performance and maintenance with descriptions of some applications that implement the ideas. Minar *et al.* [MKM99] use mobile agents to create routing tables in mobile wireless networks. In the application, mobile agents wander a network, visit mobile hosts, and exchange routing data with other agents. The implementation is straightforward and quickly and efficiently handles changes in the state of the network. It capitalizes on items three, four, and five of Lange and Oshima's list.

Baldi *et al.* [BPR98] leverage items one and two in an application that routes video-conference network traffic. The application deploys mobile agents to establish an optimized route that efficiently routes video-conference network traffic in the absence of multi-cast.

Bieszczad *et al.* [BPW98] use mobile agents for system administration and monitoring tasks. The state of the network can affect observations, so the project locates autonomous agents at trouble sites. If the network connection collapses, an agent has enough logic to handle different situations, demonstrating items four and five on the list of seven.

Muldner [Mul98] proposes using mobile agents to distribute and collect students' homework assignments at Acadia University. The plan uses mobile agents to disseminate information and install software on students' personal machines at times when system utilization is low. The structure shows evidence for items one and four from Lange and Oshima's motivations.

Points two and seven are addressed in StormCast [Joh98], a system to monitor weather in the North Sea. It deploys mobile agents to visit weather stations loosely connected by a wireless network. Agents filter data at sites and handle failures in the network.

1.4 Summary

None of the example mobile-agent applications hinge on the use of mobile agents. Each application could have been engineered without mobile-agents, but the creators argue that software mobility mitigates design and maintenance of the code base.

In this chapter we presented a rough sketch of computation markets in which we promote mobile agents and their prospective hosts participate. Each agent uses electronic currency to purchase its computational resources from the host at which it executes. In the next chapter we describe in more detail the markets as well as the system-wide benefits of market participation.

Chapter 2

Motivation

In this chapter we identify several problems faced by administrators, designers, and writers of mobile-agent applications. We see that there is little reason to host mobile agents. Realistic applications will likely involve many agents and it will be necessary for the system to stratify the agents' performance. There is also the danger that agents will exploit network resources and cause undue resource contention. Finally, agents need the ability to coordinate at a high level to distribute their load evenly across the network and over time. Market-based control can address each of these points.

2.1 Incentive

Thus far, we have yet to see a mobile-agent application operate in an open environment. We now discuss the participation limitations of broadly deployed distributed applications and conclude the section with evidence of how a market system eliminates the limitations [BKR99].

We classify the environments in which most mobile-agent systems to operate as closed because the range of sites to which an agent can jump is restricted to a small number of hosts running within a common administrative domain. It is typical for all agents in the system to only represent the interest of a single user or cooperating

group of users. Typically agents only communicate with other agents of the same application.

The number of sites to which an agent can jump is limited. Why should a host allow an agent to visit at all? Consider web browsing. In many of situations there is a clear advantage for web servers to supply their resource (information) to arbitrary clients. This information dissemination is generally done to boost the reputation of the host site's owners, clients, or products.

Computational resources exported by mobile-agent hosts are more difficult to control than the flow of information. A host site has no assurance that an arbitrary agent's actions will have any beneficial outcomes. Hence, there is little incentive for a host system to provide resources to an arbitrary agent. Not only is there the additional congestion incurred through normal mobile agent use, but there are additional risks from denial-of-service attacks and other irresponsible resource usage.

A mobile-agent system's value depends on the number of participating host sites as well as the number of participating users and agents. Most non-research mobile-agent applications assume that agents are all issued by cooperating owners. Typically, only one entity issues mobile agents for a task. This entity may be represented by many users working for a university or company, but the interests of the agents are generally complementary and it is in agents' best interest to cooperate. Essentially, agents in such situations can be viewed as having a common owner.

Even if all agents share a common goal, their use distributes decision-making processes throughout the network. To perform efficiently, agents must be able to coordinate and assess the impact of their actions. Ideally, the medium for this information exchange should be fast and incur minimal overhead.

We see few real-world examples of agents coordinating with agents involved in other applications. In stark contrast, consider the World Wide Web. A user's browser can send requests to thousands of host sites to retrieve information. Much of the in-

formation retrieval is more than simply examining an HTML file. Often browsers exchange cookies with servers, negotiate security protocols, retrieve dynamically produced web pages, download applets, and forward retrieved information to other applications to be processed. A typical web browsing-session can involve the use of several dozen application programs. We would like to be able to extend the same flexibility to computational mobility.

To overcome the limitations of contemporary mobile-agent systems, we propose to establish an economic market for computational resources and services. A mobile agent that arrives at a host site will purchase the resources necessary to complete its task. These resources could include access to the CPU, network and disk interfaces, data storage, and databases.

Presumably, agents provide valuable services to users. It is possible that other agents would also benefit from the service, so agents could sell services to users and agents. Eventually, currency accumulates at host sites. A host's administrator distributes revenues to its users, who in turn disperse their income to their agents, completing the cycle.

It is not necessary for the currency used in computational markets to be tied to legal-tender currency. If the currency is exchangeable for real dollars, however, system administrators can essentially export and import their computational resources. Access to under-utilized resources may be sold to mobile agents (resource export). If local resource contention is high, users may launch agents to carry on computation elsewhere (resource import).

Currency is scarce and budgets are finite. Since all resource consumption is tied to expenditure, each agent's lifetime is limited. The extent of a denial-of-service attack, wanton consumption done with the intention of excluding other users from the resource, is limited. If resources are fairly priced, a host will be happy to entertain a denial-of-service attack to maximize revenues. An efficient pricing policy will ensure

that demand and price are positively correlated and make denial-of-service attacks extremely expensive operations, deterring offenders.

Price, the same mechanism that discourages wasteful consumption, serves as a simple metric of resource contention and site congestion. Price advertisement provides a simple means of agent coordination as follows. Revenue maximizing hosts will charge what the market will bear. High prices due to congestion give agents incentive to distribute themselves evenly throughout the network or defer execution to a less congested time. Thus a pricing system effectively implements both temporal and spatial load balancing.

2.2 Prioritization

A mobile-agent host will accommodate many agents. In many circumstances, it is preferable to give certain agents better service than others. One agent may have a tighter deadline than another. Some agents run system-critical tasks, while others handle superfluous tasks. We have experimented with priority queues to address prioritization under simulated computing loads [BKR98a]. Under our priority queues, an agent at one priority is served first-come-first-served whenever there is no other agent present of a higher priority. We find that a host that prioritizes agents in this fashion loses control of resource allocation; the stratification of performance between agents of different priorities is determined by properties of the workload such as arrival rate and job size. Frequently, agents beyond the third or fourth level of priority do not receive service of usable quality. Stahl confirms that the marginal utilization of more than three levels of priority is rarely useful [Sta97].

If we are to provide fine-grain resource control to hosts, we must look beyond simple priority queues. Markets provide a variety of instruments to fine tune resource control. A seller can implement a variety of pricing plans, from offering a fixed price, to setting quantity or time-dependent price schedules, to holding an auction. An

authority that governs a market has additional power, too. For example a government imposes taxes to redistribute wealth and to cultivate desired consumption habits. Furthermore, monetary policy can control the value of currency which influences market participants' behavior.

2.3 Coordination

Distributed systems provide engineers with enormous computational capabilities and enable many application environments at the cost of reliance on many resources. In computational systems as usage increases, the quality of service declines. Effective planning requires that an agent can assess the cost of its actions as well as the state of resources around the network.

A market provides usage information through prices. An agent can infer the cost of its actions through the market and act appropriately. The choices that an agent makes may be to substitute one resource for another or to wait until high prices, which connote congestion, subside to access a resource. This reasoning shows that prices support both spatial and temporal load balancing. We experimented with the use of price to distribute load throughout the network [BKR98b]. We found that agents in our simulations need only to sample prices from two or three of about ten alternatives to achieve reasonable load balancing.

Schaerf *et al.* have similar experiences. The show that systems often operate more efficiently in the absence of direct agent to agent communication, but that agents must still be able to assess the state of the network [SST95]. We believe that price information can allow agents to coordinate their actions without the overhead of peer-to-peer connections or complex protocols.

2.4 Fault Tolerance

Finally we address an agent's capacity to consume resources beyond the scope of its user's or host's intents. Disconnection of a software agent implies that the agent's owner imbues the agent with autonomy. It is desirable to place limits on an agent's lifetime and capabilities and still allow an agent to reason about the cost and benefit of its actions. A market provides rational lifetimes through prices and mechanisms that an agent uses to optimize its utility. An intelligently designed agent alters its actions to react to a dynamic environment. The agent's budget constraint provides a guaranteed limit on the agent's effective lifetime. This guarantee provides a degree of fault tolerance and market reasoning allows the agent to plan how to use the available resources more effectively.

2.5 Summary

Markets smooth and distribute resource consumption over distributed systems. We believe that market-based analysis and resource-consumption policies can provide structure to distributed computing environments. In particular, mobile-agent systems have much to gain. Exchange of valuable currency motivates potential mobile-agent hosts to accommodate arbitrary mobile agents. Markets provide efficient mechanisms for prioritization, distribution of consumption, and flexible consumption limits with which an agent can reason to tune its performance.

Chapter 3

Related Work

Economics has inspired computer science researchers as far back as the 1960's. Ivan Sutherland used auctions to allocate blocks of computing time to mainframe users [Sut68]. The complexity of computing systems inspires many scientists to compare computation to both ecological and economic systems [Hub88, Cle96]. More recently, computer scientists have begun to use economics, in particular game theory, to solve traditional computational problems such as shortest-path and network-flow problems [Nis99b, Nis99a, BPS95]. Game-theoretic approaches generally do not solve problems any faster than traditional algorithmic techniques, but the abstraction of self-interested agents provides a mechanism that can be used to apply solutions to distributed environments and analyze fault tolerance.

This chapter provides an overview of how market systems have been applied to computational environments. We consider traditional system applications that use economic spirit to optimize allocation problems in operating systems and networks. Much of the promise of pricing computational resources includes a concise effective method of load balancing and improving system utilization. We summarize several projects that achieve the promise. There are several projects that use markets to regulate mobile-code systems and we cover them in this chapter as well. Finally, we look at different pricing mechanisms as well as underlying resource-management

frameworks.

3.1 Operating Systems

Many resource-allocation problems involve difficult optimization. The difficulties frequently stem from NP-hard integer-packing problems. Heuristics can be employed to optimize global system performance and economics provides good heuristics even in the absence self-interested agents.

Ferguson *et al.* [FNSY96] investigate cooperative resource allocation through market methods. They apply mathematical economics to model resource allocation in distributed systems where there is user feedback. The measurement and formalization of the allocation process allow them to construct hybrid allocation schemes that adapt to different user access patterns. For example, they propose two algorithms for data placement and analyze the cost and performance of each. Their market-based meta-allocation process weighs the cost and performance of each algorithm and to decide which one to use at run time.

Agorics, Inc. [DM88] casts system tasks such at storage management, garbage collection, and process scheduling as auctions between self-interested agents. In fact, in a multi-user operating system, many running processes do have conflicting interests. Agorics recognizes the conflict and allocates resources to agents with the greatest needs. The magnitude of an agent's needs is measured through the agent's bids for resources.

3.1.1 Memory

Harty and Cheriton [HC96] developed market-based memory allocation mechanisms. Market prices determine whether a process' data should reside in local memory, for faster access, or on the disk, where storage is more abundant. Each process has a budget management component that decides at what priority it should access memory.

The system poses an interesting paradox. A low priority process that frequently fetches data from disk imposes a performance hit on the entire system. To avoid this, Harty and Cheriton charge processes for the bandwidth incurred in paging memory from a separate monetary reserve that the agent has just for this purpose. The two monetary accounts are distinct and are not transferable.

3.1.2 File Placement

A related model uses cooperative utility maximization to place files on networked computers [KS89]. The market-influenced analysis allows both local and global search algorithms to compute file placement to minimize network latency incurred in data transmission.

3.1.3 Routing and Packet Forwarding

Schnizlein [Sch98] looks at the cost of carrying Internet traffic. A standard economic argument is to price a good at the marginal cost, but the cost is often difficult to compute practically. Additionally, usage-based billing is unpopular with consumers. Schnizlein proposes that each user subscribes to a quality-of-service level with imposed usage limits. A user that exceeds her limit is demoted to a lower quality-of-service level.

3.2 Utilization

An important factor that motivates the use of market-based resource allocation is that markets can be easily applied to distributed environments where decisions must be made with incomplete information or where the cost of obtaining information is high. Information distribution can be costly, but information shortages can lead to resource under-utilization. In this section we examine how others have applied markets to increase efficiency or performance.

An example of where information trade and cost publication can lead to higher efficiency is evident in the work of Clearwater *et al.* [CCDS96]. They allow workers at Xerox PARC to exchange heating and cooling rights used for climate control. The project constructs a thermal market and alloys employees to exchange climate control to match their preferences. The result of trade is not only are workers more content with the climate in their offices, but the system saved Xerox up to 10% of their original climate control costs on hot days and around 5% through the rest of the year. One may question whether Xerox's original climate control policy kept the building colder than the average employee preferred. It is arguable that the market simply allowed an employee to waive the right to cool an office. Regardless, the market allows users to express their preferences.

3.2.1 Load Balancing

Markets provide a means to compare the cost of alternatives. In the previous example at Xerox PARC, the costs focused on users' preferences, but costs also connote resource scarcity and quality. Traditional load balancing relies on careful measurements of resource requirements to construct, frequently simple, heuristic policies to guide workload placement. These measurements and policies are useful, but it is difficult to improve upon simple techniques and make load sharing scale to environments with used by agents with conflicting interests [ELZ86, LO86].

There have been several research projects that involve using markets to determine a program's initial execution location [FYN88, WHH+92, DM88, CMM97]. In SPAWN [WHH+92] and Challenger[CMM97], processors and users participate in auctions to distribute processes efficiently. SPAWN uses second-price auctions to avoid a user having to second guess competing demand. [1] Challenger is a simulation to demonstrate how bidders can learn through participating in repeated auctions the

[1] We discuss second-price auctions in Section 6.1.

errors in job requirement estimations.

Maheswaran *et al.* [McIB99] use local Cobb-Douglas utility functions to decide where to place and allocate priority to user programs. They show substantial improvement over first-come-first-served policies, but because of asymptotic pricing policies, the system is always under-utilized.

Mariposa [SDK+94] is a distributed database that uses markets to optimize the placement of data over the network. Sites buy and sell stored objects to speculate on the query value of the information contained by the objects. A broker performs queries for its user. The broker's budget decreases over time and reflects the user's value for timely information.

With Mariposa, Stonebraker *et al.* explore how their market system advertises prices and database capabilities. The system can guarantee access through coupons and bulk orders. There are provisions for broadcast advertisements, locally posted prices, and a yellow pages to serve as a meta-database.

3.2.2 Fault Tolerance Through Replication

In a similar fashion to Mariposa, other projects have used economic ideas to calculate the benefit of distributing data, possibly redundantly, through the network. Anastasiadi *et al.* [AKNS98] consider a model where information is replicated throughout the network at servers that sell access to data. The authors do not address how to distribute the data, but focus on the use of prices to balance the load incurred from data access. The use of load balancing substantially reduced response time in scenarios under high-frequency data access.

In a more application-specific project, Karaul *et al.* [KKO98] implement a web server that uses markets to distribute client requests among a farm of web servers. A user who desires web content contacts a well-known server that creates an agent for the user. The agent's goal is to find a server among a group of redundant web

servers that can serve the user's request quickly under a constrained budget. The idea is that prices convey load-balancing information and that agents can serve requests with minimal communication.

In both Anastasiadi's and Karaul's projects, agents compete for resources, but the research assumes that agents have homogeneous goals and that an agent has no alternative to market participation. The systems use economics to provide mechanisms for information exchange in distributed systems, but do not exhibit competitive behavior among rational agents.

MarketNet [YDFH98] relaxes the participation and cooperation assumptions to use pricing systems to protect resources. The idea is that all resource access has an associated price, possibly in different currencies, and that it is not feasible for an untrusted party to accumulate enough savings to access sensitive resources. The project does not address the possibility of several parties pooling their resources to access information that otherwise would be unavailable with a smaller budget. The conspirators could them replicate the information. MarketNet also relies on the ability to create multiple currencies that complicate planning since there would no longer be a single currency to use as a cost metric.

3.2.3 Business Ventures and Web Projects

Recently, several companies adopted market-based computational ideas to sell computational services. Generally, the idea is to compensate people for remote access to their home or office computers. In 1997, distributed.net[LBG$^+$] asked Linux users to download a client that would access key blocks to decrypt a message encrypted by RSA Security Inc. [2] The motivation for distributed.net was to show that brute force could break 56-bit encryption. The project took 250 days to complete. In exchange for cycles, the owner of the computer to locate the secret key won a prize of $2,000.

[2] `http://www.rsa.com`

RSA's total outlay towards the prize was $10,000, of which distributed.net received $2,000 for supplying the computational infrastructure.

Several companies have adopted distributed.net's ideas as a business model to use under-utilized personal computers to attack large corporate and academic computational problems. Companies like XDegrees,[3] United Devices,[4] and Entropia[5] distribute free clients that implement distributed programming environments. The compensation for use varies between companies. Some enter users into lotteries for substantial travel packages ($25,000). The number of lottery entries a user receives depends on the speed of the user's computer and usage hours. Other companies attract users by assuring that participation will further cancer research.

Mojo Nation[6] is a peer-to-peer resource sharing environment. Its primary use is to share files among networked computers. The cost of storage is represented with an imaginary currency called "mojo" that users trade to access shared files and information services. The system supports publication and reputation look-up services.

Enron Communications[Joh99] sells contracts that guarantee network bandwidth to large corporations. In 1999, Enron Communications allowed its clients to sell unused monthly portions of bandwidth along a network link between New York and Los Angeles and extended the market by adding bandwidth contracts connecting New York and London in 2000. The company established a market for bandwidth that its clients can monitor in real time. With the flexibility of a bandwidth market, a company can access and pay for high bandwidth communication only when it needs it, say for occasional video conferences.

[3] `http://www.xdegrees.com`
[4] `http://www.uniteddevices.com`
[5] `http://www.entropia.com`
[6] `http://www.mojonation.net`

3.3 Mobile Agents

We will now look at two areas of existing work directly related to mobile agents. In particular we are interested in using mobile agents to further fault tolerance and in the use of markets to regulate mobile-agent systems.

3.3.1 Fault Tolerance

Mobile agents provide an intuitive means to distribute computation throughout the network. An agent makes choices to route its computation through the network. It is possible that an agent will fail after relocating to a new site. Mohindra *et al.* [MPT00] explore the possibility of recording an agent's actions. Upon failing at a site, the agent's last successful known state is cloned and the agent randomly chooses a new site alternative.

Shehory *et al.* [SSCJ98] investigate the possibility that an agent clones itself to improve its chances of completion. An agent that operates on an overloaded host considers locating a clone at a remote machine as well as dividing tasks between the original and cloned agents. The technique improves system utilization by providing metrics for distributing computation as well as allowing an agent to avoid poor performance while stuck at an overloaded or faulty host.

3.3.2 Markets

A consequence of agents' ability to autonomously migrate is that it may be difficult to locate an errant agent or ensure that the agent does not run forever. A market establishes flexible quotas for an agent. This idea has been used in mobile-agent systems from the beginning. Telescript[Whi96], one of the earliest mobile-agent systems, has provisions for agents to carry resource-access permits. The power of a permit diminishes over time, travel, and use. Roughly, a permit is currency that represents an agent's potential to consume, though exchanging permits is not as general as a

market since a permit for one resource is not easily exchangeable for another; permits are not fungible.

Within Geneva Messengers[Tsc97], another early mobile-agent system, there exists a market for computation as well as memory. Prices for each commodity vary with demand and an agent's endowment decreases over time. A shortcoming with the Messengers market is that agents cannot reason about the impact of an action on the agent's run-time or performance. There is no direct mechanism for an agent to assess its resource access or request a different priority. In a sense, Messengers' market is an elaborate accounting and quota system.

While not general mobile-agent systems, POPCORN [RN98] and JavaMarket [AAB98] provide implementations that distribute Java programs to under-utilized machines. Users submit their machines and programs to a market that pairs up and distributes programs to host machines. A host collects a fee determined by one of several types of auctions prior to accepting the program. The applications run under POPCORN and JavaMarket are computationally intensive since the sandboxed applet-like run-time environment prevents most inter-process communication.

To support a method for untrusting agents to meet within the IBM Aglets mobile-agent system, researchers developed e-Marketplace [Miy], a bazaar where agents can discover and exchange information and services. The framework ignores computational resource allocation, but is useful for resource and service discovery.

3.3.3 Planning

A mobile agent assembles an itinerary of hosts to visit. Moizumi [Moi98] applies traveling-salesman problem (TSP) to optimize a mobile agent's performance. TSP requires an agent to visit a set of hosts, in an unspecified order, traveling the minimal distance. In the mobile-agent environment, the appropriate "distance" metric is time. The itineraries about which we reason in Chapter 4 are less general in that they

assume a fixed order, but allow agents to choose to visit one of several hosts to complete a task.

Our problem then becomes one where the agent attempts to find a shortest path in a graph where the edges represent both network transfer times as well as task completion times. The nodes represent decisions that the agent may take, either destinations or the priority at which to compute. In real-world applications, the weights, or latencies, of the edges vary over time. Wellman *et al.* [WFL95] consider planning under conditions where distances are uncertain in the stochastic shortest-path problem.

3.4 Pricing

A substantial portion of market design and participation concerns price computation. Gupta *et al.* [GSW97b, GSW97a] and Paschalidis and Tsitsiklis [PT98] consider adaptive pricing algorithms where sellers independently attempt to maximize revenue through price computation.

Centralized allocation is also possible. Cheng and Wellman [CW98] present a tatonnement algorithm that attempts to find equilibrium prices and allocation among many buyers and sellers. The algorithm searches over allocations to minimize excess demand by iteration over prices that sellers post and quantities that buyers reveal. It is guaranteed to find a solution when market participants have convex utility functions and there is gross substitutability between goods– goods are not complementarities for one another. In practice, it frequently converges to a solution when gross substitutability does not hold, however.

3.4.1 Auctions

An auction is type of market where buyers and sellers submit competing bids and offers. The market type is often employed to allocate resources when communication

costs are high or when information is not readily available to one party. Wurman *et al.* [WWW01] provide a good overview of the characteristics of many types of auctions and implement them electronically using the Michigan AuctionBot [WWW98]. A more commercial example of electronic auction use is eBay[7] where a seller can, for a fee, hold a customized auction.

Selling a good whose value is independent of other allocations is usually not difficult. There are many cases, however, where the value of one holding depends on another. To efficiently allocate multiple goods one can use a combinatorial auction, an auction where the auctioneer sells bundles of goods. There have been a flurry of papers addressing efficient computation of revenue optimization in combinatorial auctions. Leyton-Brown *et al.* [FLBS99] and Sandholm [San99] use heuristics to quickly search through bids that represent bundles of goods. Their frameworks allow a buyer to bid for mutually exclusive bundles. For example, an agent may be interested in goods A and B, or good C, but not the entire triple. In iBundle [Par99], Parkes attacks the dual of the allocation problem to compute an approximate solution to the generalized Vickrey auction. He shows that it is unlikely that an agent should deviate from a truthful policy when the auctioneer approximates the generalized Vickrey auction closely enough.

Another method of allocating multiple goods with complementarities is to hold a sequence of auctions. Boutilier *et al.* [BGS99] show that it is a user's optimal strategy to reveal her preferences to an agent that bids a sequence of auctions under the following conditions. All users must use the same agent type. Each agent repeatedly participates in sequences of auctions to learn strategies that maximize its user's utility. The final allocation is determined by the result of the last sequence of auctions in which the agents participate.

SmartAuctions [MMV95] were proposed as a solution to prioritizing network traf-

[7] `http://www.ebay.com`

fic. Users endow packets with currency. At a router, the packet participates in an auction for priority. The problem arises, however, in that a user's utility does not concern whether one packet arrives, but on an entire sequence's arrival. There is no clear strategy on how a packet should participate in an auction given the utility dependence upon other packet transfers in a message.

3.5 Resource Management

In this dissertation, we propose that markets can effectively regulate distributed systems. Effective regulation requires usage enforcement, so before we proceed further with creating resource access policies, we look at what level of resource control is possible in existing systems.

Within D'Agents [Gra97], an early mobile-agent system, agents have absolute usage limits and file-access permissions. By itself D'Agents has no support for prioritizing agents' performance, but Cooper and Gray [CG00] provide an extension of D'Agents running under the Q-Linux operating system that prioritizes each agent's CPU access.

In a similar spirit, resource control can be conducted within the Java Virtual Machine (JVM). J-Res [CvE98] is a package that allows fine-grained resource control for applications running under J-Kernel, a modified JVM. More recently, Sun Microsystems incorporated the Java Virtual Machine Profiling Interface, JVMPI, as part of the Java specification. The Java Usage Monitor, JUM, [Lou99] is a Java-based package that allows resource control to Java programs running within modified JVMs running on top of Solaris or Linux. Packages like J-Res and JUM have the drawback that their byte-codes are not portable to other JVMs, or even each other.

The JVM is a user-level program and accurate resource control relies on the operating system upon which the JVM runs. Alta and GVM [BTS+98] are operating systems developed at the University of Utah that allow fine-grained CPU and net-

work access scheduling and process-level memory control. The differentiating factor between the two operating systems is the design structure; Alta is based upon a micro-kernel architecture, while GVM has the traditional monolithic design. Both are based heavily upon JVM specifications and can only run Java byte-code programs. Neither system performs significantly faster than JVM-internal allocation mechanisms such as J-Res.

Further extraction from process-level to operating-system level resource control leads to the idea that the network should have common resource control and access. Distributed systems such as Xenoservers [RPM+99] and Grid systems like Nimrod/G [ASGH95, BAG00] support network-wide resource control. Xenoservers is an architectural model that provides an environment for execution and resource-accounting of untrusted code. Under Xenoservers, a user specifies resource-usage needs and Xenoserver guarantees and schedules those needs if the user can be accommodated. The architecture supports currency-based accounting procedures that bill a client for resource congestion. Nimrod/G is a distributed system where applications can be launched and scheduled among many node machines. The work supports real-time scheduling as well as market-based methods for resource accounting. Under Nimrod/G, scheduling is coordinated by a centralized dispatcher.

Chapter 4

Application Model

In this chapter, we construct a model of the jobs that mobile agents execute and show how we model the computation required to finish the jobs. We present notation common to Chapters 5-8. At the end of this chapter and Chapters 5-8, we assemble tables of the notation we introduce in the respective chapters. We compile all the notation in Appendix A.

We consider an environment where each application has a sequence of jobs to execute. The job sequence and the nature and size of each job are known at the time of an agent's creation. We call a set of jobs and location choices an itinerary. In our simulations M_i, the number of jobs that comprise the i-th agent's itinerary, is exponentially distributed and $q_{i1} \ldots q_{iM_i}$, the sizes of the individual jobs, are Pareto distributed.

It is reasonable to question the validity of assuming that applications can estimate their resource usage. Narayanan *et al.* apply history-based prediction techniques to estimate computation time and electrical power consumption of downloading, rendering, and displaying JPEG and three dimensional images on mobile computers [NFS00]. They find strong relationships between the amount of computation and energy to uncompress, render, and display images with the image display quality parameters. In Section 5.5.4, we investigate the effect of error on an agent's job-size

estimation inside our model.

Each job may be completed at one of several host sites. At each host that an agent visits, it must purchase the computation consumed by its job. At the j-th site that the agent visits, the agent receives a portion, x_{ij}, of the host's computing capacity, c_{ij}. The proportion of computation that the agent receives depends upon the host's allocation policy and decisions taken by the agent in negotiating access to computation.

The allocation mechanism is the medium that an agent uses to express its preferences to the host (the principal) to negoiate resource access. A *mechanism* is a function that maps agents' actions to outcomes. Chapters 5 and 6 present two mechanisms derived from different agent and host motivations.

Under each mechanism, the time taken for the i-th agent to complete its j-th job is simply, $q_{ij}/c_{ij}x_{ij}$, the size of the agent's job divided by the absolute amount of computation the agent receives.

The choice of execution location is important because hosts have varying capacity and congestion, both of which we assume are common knowledge among agents. The choice of a set of hosts that optimizes the end-to-end latency of an agent's itinerary is a problem that we consider in Chapter 7. Once the agent relocates to a site, we assume that the agent commits to finishing its next job at the site. While at the site the agent must determine at what priority to execute. Obviously there are higher costs associated with high priority execution.

The host deducts the cost of execution of the i-th agent's j-th job, t_{ij}, the agent's endowment, E_i. The user determines the endowment at the agent's creation. The endowment expresses the agent's priority and its user's preference that the agent completes quickly. An agent with an important or larger itinerary is characterized with a larger endowment. This gives the agent greater flexibility to negotiate its resource access privileges with the hosts it visits. We model the quotient of an agent's

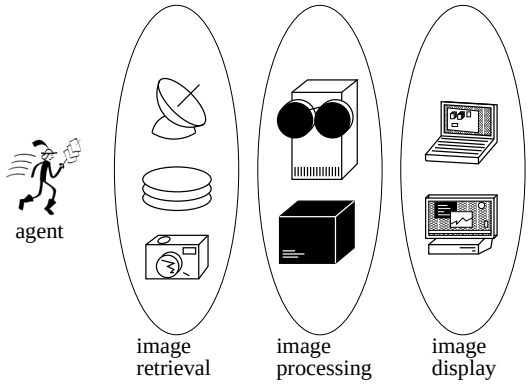

Figure 4.1: An example mobile-agent itinerary choices. The agent must visit one host from each group and choose at what priority to execute.

endowment and its job-size sum as a positively truncated Gaussian random variable to reflect users' differing tastes and relative status.

Figure 4.1 shows an example of a mobile agent's possible itinerary. The agent needs to retrieve an image from one of three sites, process it at one of two other sites, and finally present the results to its owner. Each site will contribute different latencies to the agent's total performance. The latency incurred by a visit stems from the host's ability to carry out the agent's computation as well as the agent's ability to negotiate price and performance at the host in the midst of other agents attempting similar jobs.

Table 4.1 lists the names and definitions of variables that we use to describe agents and their itineraries in future chapters. In the table, we use subscripts to denote context of which agent or host that we refer, though we will drop appropriate subscripts when we discuss an isolated agent or host.

c_{ij}	The capacity in computational units per second that the j-th host visited by the i-th agent can sustain.
E_i	The i-th agent's endowment.
i	The agent index.
j	The job index.
M_i	The number of jobs that the i-th agent must perform.
N_{ij}	The number of agents at the j-th host that the i-th agent visits.
q_{ij}	The size of the i-th agent's j-th job.
t_{ij}	The monetary transfer from the agent to the host for computation of the i-th agent's j-th job.
x_{ij}	The fraction of capacity for the j-th job that the i-th agent receives.

Table 4.1: Notation used to describe agents and their itineraries.

Chapter 5

System Utilization

We first apply markets to regulate a simple model where users collectively own the computing resources [BMI+00]. The users would like to have high resource utilization, efficient execution, and the ability to prioritize their jobs. To this end, a user endows its agent with currency that reflects its priority with respect to other agents in the system. The agent's objective is to complete its set of jobs as quickly as possible while respecting its budget constraint. To simplify our resource-allocation policy, we assume that an agent does not have the ability to transfer its endowment back to its user, and thus there is no incentive for an agent to save.

In the context of an environment where users share ownership of the network resources, the savings restriction is not catastrophic. All currency spent by agents is eventually returned to users through the system's higher-level priority-allocation policy. This policy, perhaps, is determined by the system administrator who decides which users are important and the amounts of system resources that each user can obtain.

An agent has an itinerary that we describe in Chapter 4, and each host that the agent visits presents the agent with the allocation mechanism defined in Algorithm 1. The j-th host runs the algorithm every time an agent arrives or departs. The variable θ_j is the published rate at which the host currently collects revenue. An agent's bid,

Algorithm 1 Proportional payment allocation mechanism for use by the j-th host

1: **if** $N_{ij} = 1$ **then**
2: $\quad x_{1j} \leftarrow 1$ {no congestion}
3: $\quad u_{1j} \leftarrow 0$
4: **else**
5: $\quad$ Agents simultaneously submit $g_i()$ to host.
6: $\quad \theta_j \leftarrow \sum_{i \in [1 \ldots N_{ij}]} g_{ij}(\theta_j) \;\; \theta_j \in (0, \infty)$
7: $\quad$ **for each** $i \in [1 \ldots N_{ij}]$ **do**
8: $\quad\quad x_{ij} \leftarrow g_{ij}(\theta_j) / \sum_{k \in [1 \ldots N]} g_{kj}(\theta_j)$ {allocate proportionally to payment}
9: $\quad\quad u_{ij} \leftarrow g_{ij}(\theta_j)$
10: $\quad$ **end for**
11: **end if**

$g_{ij}()$, obligates the i-th agent to pay the host at a rate of $u_{ij} = g_{ij}(\theta_j) \leq \theta_j$ conditioned that the population of agents collectively pays the host a rate of θ_j for all available computation. In return, the i-th agent receives a portion of computation equal to $x_{ij} = g_{ij}(\theta_j)/\theta_j$. Because $\theta_j = \sum_{i \in [1 \ldots N]} g_{ij}(\theta_j)$, every agent pays the same price per unit of computation and receives an amount of the resource exactly proportional to what it pays. As a special case, when there is no competition for computation, the price of computing is zero.

An agent's bid function is not an arbitrary function. It must have the properties

- $\partial g_{ij}/\partial \theta_j \leq 1$;
- $g_{ij}(0) = 0$;
- $\partial g_{ij}/\partial \theta|^{\theta=0} = 0$;
- $g_{ij}(\theta_j)$ must be twice differentialable on $[0, a]$;
- $\theta \in [0, a], a > 0 \;\; \partial^2 g_{ij}/\partial \theta^2 < 0$;
- $0 = g_{ij}(a)$;
- and $0 = g_{ij}(b) \;\; \forall b > a$.

The conditions that describe the bidding function at the origin state, in the absence of competition, the agent bids nothing and receives all of the available computation. As competition increases, concavity and the limited slope of the bidding function assures that the agent necessarily receives a smaller portion of the computation and that the agent never attempts to bid for more than 100% of the available resources. The behavior on the range $[a, \infty]$ guarantees that the agent has a simple, well-defined maximum expenditure rate for computation.

In implementation, we describe an agent's bidding function as a real-valued triple, $(\alpha_i, \beta_i, \gamma_i)$, that we derive in the next section.

5.1 Agent Objectives

We assume that all agents in our system have similar utility functions that reward low end-to-end latency. Without loss of generality, let us assume throughout the rest of the chapter that each agent is visiting its first host. The agents need not have the same itinerary length, however. The utility function for the i-th agent with M_i remaining jobs is the remaining latency, that is,

$$U_i = \sum_{j=1}^{M_i} \frac{q_{ij}}{c_{ij} x_{ij}} \tag{5.1}$$

and the agent's goal is to minimize the utility function through its choice of u_{ij}, the rate at which it spends its endowment at the j-th host. For now, we ignore latencies involved in transporting an agent from one host to another. We assume that the agent has already chosen its route and we explore how the agent chooses its route in Chapter 7.

The agent minimizes U_i under its budget constraint. Since the minimization program respects the budget constraint, our derived strategy is guaranteed to produce a budget that bids only if the agent can complete its itinerary given its congestion

estimations of future hosts. The minimization program is

$$\begin{aligned} &\min_{u_{ij}} \sum \tau_{ij} \\ &\text{s.t. } E_i \geq \sum_{j=1}^{M_i} \tau_{ij} u_{ij} \end{aligned} . \tag{5.2}$$

The time taken for each job is

$$\tau_j = \frac{q_{ij}\theta_j}{c_{ij}u_{ij}} \tag{5.3}$$

and the agent pays the j-th host an amount equal to the product of the rate u_{ij} it agreed to pay in the allocation mechanism and the time, τ_{ij}, taken to perform the job.

5.2 Optimization

In this section we derive a bidding strategy for the mechanism defined by Algorithm 1. The strategy will explicitly define how an agent should bid to optimize its utility and implicitly define a budget for the agent.

Our strategy guarantees that an agent will be able to complete its itinerary as quickly as possible in the absence of knowledge of other agents' strategies. The only assumptions we make on planning is that the agent knows the size of all of the jobs, q_{ij}, that comprise the itinerary; an expectation of the level of congestion at each host, θ_j; and the capacity, c_{ij}, of every host to be visited. We show that in absence of network state change, the optimal strategy holds the agent's expenditure rate fixed. Additionally, the allocation that results from multiple agents' bid submission is unique due to the convexity of each bidding function.

5.2.1 Simplification

To attack the problem, we simplify the problem slightly. We consider how the agent may optimize its utility as a function of the competing agents' contribution to con-

gestion. The new domain for the agent's bid is

$$\theta_j^{-i} = \theta_j - u_{ij}. \tag{5.4}$$

The agent's utility is a convex function of u_{ij}. A common method of optimizing convex functions under convex constraints is to apply Lagrangian relaxation [Lan87]. The Lagrangian for the agent's utility function is

$$\mathcal{L} = \sum_{j=1}^{M_i} \tau_{ij} + \lambda \left(\sum_{j=1}^{M_i} \tau_{ij} u_{ij} - E_i \right). \tag{5.5}$$

The parameter λ is called the Lagrange multiplier. Its value reflects the agent's price of computation. The difference by which we multiply λ is the agent's constraint that it spends no more currency than it has, E_i. The optimization of the objective function in the presence of the constraints is identical to optimization of the Lagrangian in Equation 5.5. We compute the optimum by deriving the partial derivatives of $\mathcal{L}$ with respect to u_{ij} and λ and solving for u_{ij} when the derivatives are equal to zero. After we substitute Equation 5.3 into Equation 5.5, the partial derivatives are

$$\frac{\partial \mathcal{L}}{\partial u_{ij}} = -\frac{q_{ij}\theta_j^{-i}}{c_{ij}u_{ij}^2} + \lambda \frac{q_{ij}}{c_{ij}} \tag{5.6}$$

and

$$\frac{\partial \mathcal{L}}{\partial \lambda} = \sum_{j=1}^{M_i} \tau_{ij} u_{ij} - E_i. \tag{5.7}$$

When $\partial \mathcal{L}/\partial u_{ij}$ is zero, we solve for λ to obtain

$$\lambda = \frac{\theta_j^{-i}}{u_{ij}^2}. \tag{5.8}$$

We know at optimality $\partial \mathcal{L}/\partial u_{ij} = \partial \mathcal{L}/\partial u_{ik}$, for all pairs j and k, and using Equation 5.6, we find that there is the relationship between any two bids

$$u_{ik} = u_{ij}\sqrt{\frac{\theta_k^{-i}}{\theta_j^{-i}}}. \tag{5.9}$$

To determine the bid at the first host, we substitute Equation 5.9 for all k not equal to one into Equation 5.7 when $\partial\mathcal{L}/\partial\lambda$ is zero and solve for u_{i1} to arrive at the bidding function $f_i()$ in terms of *other* agents' bids,

$$u_{i1} = \max\left\{ f_i(\theta_1^{-i}) := \frac{\alpha_i - \beta_i\theta_1^{-i}}{\beta_i + \frac{\gamma_i}{\sqrt{\theta_1^{-i}}}}, 0 \right\} \tag{5.10}$$

where α_i, β_i, and γ_i are shorthands for

$$\alpha_i := E_i - \sum_{j\neq 1}^{M_i} \frac{q_{ij}}{c_{ij}}\theta_j^{-i}, \tag{5.11}$$

$$\beta_i := \frac{q_{i1}}{c_{i1}}, \tag{5.12}$$

and

$$\gamma_i := \max\left\{ \sum_{j\neq 1}^{M_i} \frac{q_{ij}}{c_{ij}}\sqrt{\theta_j^{-i}}, \epsilon \right\}. \tag{5.13}$$

The parameter α_i represents the agent's ability to pay for the current job given its estimate of future consumption. If α_i is less than zero, $f_i()$ will be negative, but we require that the agent's bid is non-negative. Hence if α_i is negative, the agent cannot bid and must wait until congestion subsides. We also require that β_i and γ_i are positive. When the agent has only one job left to complete, the computation of γ_i would be zero. We see later that zero-valued γ_i causes a singularity, so we approximate the outcome with an arbitrarily small value, ϵ.

5.2.2 Generalization

Every agent has a different domain for $f_i()$, so, by itself, $f_i()$ is not useful to the host. We must transform the agent's bidding function to one that includes its own bid. To

derive a bidding function that operates on the domain of all agents' bids, we take the bidding strategy in Equation 5.10 and into it substitute the definition of θ_1 in terms of θ_1^{-i} and u_{i1},

$$\theta_1 = \theta_1^{-i} + u_{i1}. \tag{5.14}$$

Over the range where $f_i()$ is positive, the substitution yields

$$u_{i1} = \frac{\alpha_i - \beta_i(\theta_1 + u_{i1})}{\beta_i + \frac{\gamma_i}{\sqrt{\theta_1 + u_{i1}}}}, \tag{5.15}$$

which yields the quadratic in terms of u_{i1},

$$0 = \gamma_i^2 u_{i1}^2 + (\alpha_i - \beta_i\theta_1)^2 u_{i1} - \theta_1(\alpha_i - \beta_i\theta_1)^2. \tag{5.16}$$

We solve the quadratic equation for u_{i1} and again restrict the range to positive values to yield

$$u_{i1} = g_i(\theta_1) := \max\left\{ \frac{(\alpha_i - \beta_i\theta_1)^2}{2\gamma_i^2} \left(-1 + \sqrt{1 + \frac{4\gamma_i^2\theta_1}{(\alpha_i - \beta_i\theta_1)^2}} \right), 0 \right\}. \tag{5.17}$$

The function $g_i(\theta_1)$ is positive over the domain $\theta_1 \in (0, \alpha_i/\beta_i)$. When θ_1 exceeds α_i/β_i, the agent's execution pauses until congestion subsides and the price drops. Figure 5.1 demonstrates an example agent bid. Increased endowment has the effect of scaling the bid function. Increasing the agent's future load makes the function's curvature softer and decreases the range over which the agent may bid. Decreasing the agent's future consumption makes the bid function sharper and more closely approximate the function $g_i(\theta_1) = \theta_1$ along the interval $\theta_1 \in [0, \alpha_i/\beta_i]$ and zero elsewhere. In the limit as ϵ in Equation 5.13 approaches zero, an agent with only one job left submits a bid function that looks increasingly like a right triangle whose hypotenuse has a slope of one and projects out from the origin to the point over α_i/β_i.

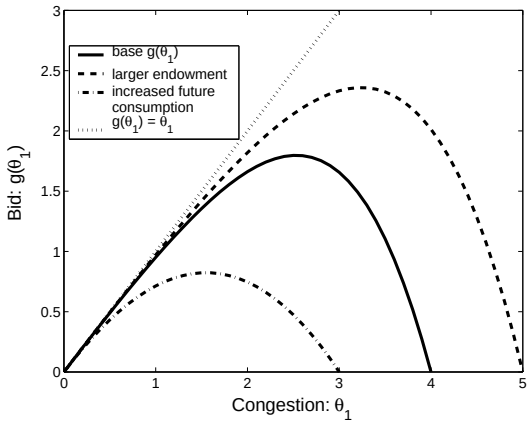

Figure 5.1: An example bid for an agent and a demonstration of how the bid changes with the agent's workload and endowment.

5.2.3 Time Invariance

We now show that it is optimal for the agent to continue to pay the host at a rate of $g_i(\theta_1)$ so long as its future load estimations do not change. This quality is important because it implies that the optimal strategy need not be recomputed in the absence of network state change.

Theorem 1 *The bidding strategy represented by $g_i(\theta)$ is time invariant.*

Proof of Theorem 1

Assume that the i-th agent submits a bid function to the host. Let h_1 be the rate at which the agent initially pays the host. The host collects currency from other agents at a rate of $\theta^{-i'}$. Let h_2 be the agent's updated bid after computing for τ time units at a congestion level of $\theta^{-i'}$.

The agent's endowment decreases by

$$\Delta E = \tau h_1 \tag{5.18}$$

and the job size decreases by

$$\Delta q_1 = \frac{\tau c_1 h_1}{h_1 + \theta^{-i'}} \tag{5.19}$$

after executing for τ time units. From Equations 5.10, 5.11, and 5.12, the agent's new bid, in terms of θ^{-i} is

$$h_2 = \frac{\alpha_i - \tau h_1 - \theta^{-i}\left(\beta_i - \frac{\tau h_1}{\theta^{-i'}+h_1}\right)}{\beta_i - \frac{\tau h_1}{\theta^{-i'}+h_1} + \frac{\gamma_i}{\sqrt{\theta^{-i}}}}. \tag{5.20}$$

The partial derivative of h_2 with respect to τ indicates how the agent's bid changes over time. If congestion for the resource does not change, then θ^{-i} equals $\theta^{-i'}$ and the partial derivative is

$$\frac{\partial h_2}{\partial \tau} = \frac{h_1}{\theta^{-i}+h_1} \frac{\alpha_i - \beta_i(\theta^{-i}+h_1) + \frac{h_1\gamma_i}{\sqrt{\theta^{-i}}}}{\left(\beta_i - \frac{\tau h_1}{\theta^{-i}+h_1} + \frac{\gamma_i}{\sqrt{\theta^{-i}}}\right)^2}. \tag{5.21}$$

If we substitute Equation 5.10 for h_1, the numerator in Equation 5.21 is zero. Hence the agent's bid does not change over time in the absence of state changes. If all agents follow the strategy described by $g_i(\theta)$, and hence $f_i(\theta^{-i})$, there is no need to recompute the allocation unless the state of the network changes. $\diamondsuit$

Theorem 1 implies that an agent does not benefit from changing its bid over the course of its lifetime unless the network state changes. The constancy is especially important in light of the fact that the agent does not know the response of its competition prior to allocation. After allocation, the agent will not wish to change its bid given the response of the competition.

5.2.4 Concavity

While the bidding strategy defined in Equation 5.17 appears complicated, it is qualitatively well behaved. We now show that the strategy is concave on the domain, $(0, \alpha_i/\beta_i)$, the only domain upon which the agent submits a positive bid.

Theorem 2 *The bidding strategy represented by $g_i(\theta)$ is concave over the interval* $\theta_1 \in [0, \alpha_i/\beta_i]$.

Proof of Theorem 2

We begin by simplifying the agent's bid function, $g_i(\theta_1)$, from Equation 5.17

$$g_i(\theta_1) = \frac{zw}{2\gamma_i^2} \tag{5.22}$$

where

$$z := \alpha_i - \beta_i\theta_1, \tag{5.23}$$

$$w := -z + \sqrt{y}, \tag{5.24}$$

$$y := z^2 - bz + \alpha_i b, \tag{5.25}$$

$$b := 4\gamma_i^2/\beta_i. \tag{5.26}$$

To prove that $g_i(\theta_1)$ is concave, we must prove that $d^2g_i/d\theta_1^2$ is at most zero, where

$$\frac{\partial^2 g_i}{\partial\theta_1^2} = \frac{-2\beta_i\,\partial w/\partial\theta_1 + z\,\partial^2 w/\partial\theta_1^2}{2\gamma_i^2}. \tag{5.27}$$

We notice that

$$\begin{aligned} \tfrac{\partial w}{\partial z} &= \tfrac{2z-b}{2\sqrt{y}} - 1 \\ &= \tfrac{2z-b-2\sqrt{y}}{2\sqrt{y}} \end{aligned} \tag{5.28}$$

and for $\partial w/\partial z$ to be zero it must be the case that

$$(2z - b)^2 = 4y, \tag{5.29}$$

which implies that $b = \alpha_i$, but this is not possible since $g_i(0)$ is zero, and $b = \alpha_i$ implies that $w\big|^{b=\alpha_i} = g_i(0) = 0$. Thus w has no inflection points along the domain $z \in [0, \alpha]$ and $\partial w/\partial z$ must be either strictly positive or negative.

We observe that

$$w|^{z=0} = \sqrt{\alpha_i b} \tag{5.30}$$

$$w|^{z=\alpha_i} = 0, \tag{5.31}$$

so $\partial w/\partial z$ must be negative, which implies that $\partial w/\partial \theta_1$ is positive.

Finally, since $\partial^2 z/\partial \theta_1^2$ is zero, we have

$$\frac{\partial^2 w}{\partial \theta_1^2} = \frac{\partial^2 w}{\partial z^2}\frac{\partial z^2}{\partial \theta_1^2} = \frac{-4\alpha_i b - b^2}{4y^{3/2}}\beta_i^2 \leq 0. \tag{5.32}$$

We combine the note that since $\partial w/\partial \theta_i$ is positive and $\partial^2 w/\partial \theta_i^2$ is negative, Equation 5.27 implies that $\partial^2 g_i/\partial \theta_1^2$ is negative. Therefore the agent's bidding function is concave over the interval θ_1 in $[0, \alpha_i/\beta_i]$. $\diamondsuit$

We will use Theorem 2 to prove that our allocation policy produces a well-defined result.

5.3 Allocation

A host executes Algorithm 1 to determine each agent's allocation whenever an agent arrives or departs the host. The process searches for a value of θ_1 that satisfies all agents' bidding functions. Satisfaction is equivalent to

$$\theta = \sum_{i=1}^{N_{ij}} g_i(\theta_j). \tag{5.33}$$

Figure 5.2 plots an example of the sum of agents' bid functions, $g_i(\theta_1)$. The abscissa represents congestion and the ordinate represents competition for computation. Satisfaction of all bid functions represents a point where the summed bid functions crosses the line with slope one — where congestion equals competition. There is always the trivial solution — where zero congestion yields zero competition.

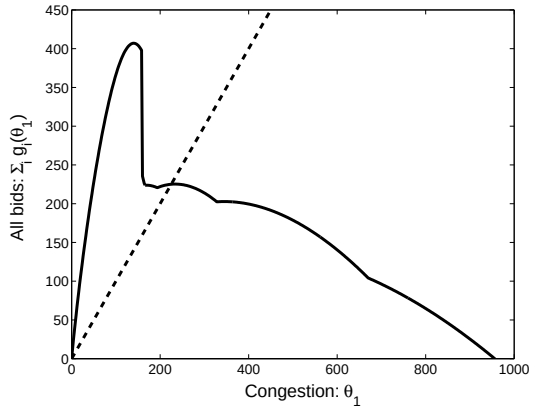

Figure 5.2: An example of an agent population's response to the host picking a payment rate. The only equilibrium points are at two intersections of the agents' response, denoted by the curve, and the line $\theta_1 = \sum_i g(\theta_1)$.

5.3.1 Uniqueness

We now show that there is exactly one non-trivial solution when there are multiple agents competing for computation at a host. We leverage the results of Theorem 2 and the properties that each bid function extends from the origin with a slope of one.

Theorem 3 *There is a unique non-trivial allocation that satisfies all $g_i(\theta)$ submitted to the host.*

Proof of Theorem 3

From inspection of Equation 5.17, the value of $g_i(0)$ is zero and $g_i()$ is continuous. The derivative of $g_i(\theta_1)$ with respect to θ_1 is

$$\frac{dg_i}{d\theta_1} = \frac{2\beta_i(\alpha_i - \beta_i\theta_1)}{2\gamma_i^2} + \frac{-2\beta_i(\alpha_i - \beta_i\theta_1)^2 + 2\gamma_i^2(\alpha_i - \beta_i\theta_1) - 4\beta_i\gamma_i^2\theta_1}{2\gamma_i^2\sqrt{(\alpha_i - \beta_i\theta_1)^2 + 4\gamma_i^2\theta_1}}. \tag{5.34}$$

When θ_1 is zero,

$$\left.\frac{dg_i}{d\theta_1}\right|_{\theta_1=0} = \frac{1}{2\gamma_i^2}\left[2\beta_i\alpha_i + \frac{-2\beta_i\alpha_i^2 + 2\gamma_i^2\alpha_i}{\sqrt{\alpha_i^2}}\right] = 1. \tag{5.35}$$

Without loss of generality, we order the agents in decreasing order of the extent of the congestion under which they will bid,

$$a := \frac{\alpha_i}{\beta_i}. \tag{5.36}$$

We define

$$G(\theta, k) := \sum_{i=1}^{k} g_i(\theta) - \theta \tag{5.37}$$

to be the excess demand of the first through k-th agents' response to a proposed congestion level θ. Note that the response is increasing as a function of k on the domain of θ in $[0, a_k]$. For the remainder of the domain, $[a_k, a_1]$, the response is constant as a function k. The value of $G(\theta, 1)$ is strictly negative for θ greater than zero.

We now show that for any value of k in $[2 \dots M_j]$, there is a unique value of θ_j greater than zero such that $G(\theta_j, k)$ is zero, by induction on k.

Base Case: *There is a unique positive θ such that $G(\theta, 2) = 0$.*

Let θ' be the smallest value that eliminates excess demand for the first and second agents. Because the sum of the slopes of the agents' bid functions extending from the origin is greater than one, and the sum of the two concave bid functions is also concave, $\partial G/\partial\theta|^{k=2,\theta=\theta'}$ is less than one. Furthermore, the slope of the excess demand is decreasing over the domain $[\theta', a_2]$. Hence, $G(\theta, 2)$ is negative along that interval. Along the interval $[a_2, a_1], G(\theta, 2)$ is equal to $g_1(\theta)$ and the slope of $g_1(\theta)$ is less than one. So $G(\theta, 2)$ must be negative on the domain $[a_2, \infty)$. Therefore, there is a unique positive θ that eliminates excess demand of two agents.

Induction Hypothesis: *For $k \leq K$ there is a unique positive θ such that $G(\theta, k) = 0$.*

Let θ_K be the value that satisfies $G(\theta_K, K) = 0$. If a_{K+1} is less than or equal to θ_K, then the theorem is proved; the bid of the $K+1$-th agent contributes only in the

region in which there was excess demand for the initial K agents.

Now let us consider the possibility that a_{K+1} is greater than θ_K. On the domain $[a_{K+1}, a_1]$, the value of $G(\theta, K+1)$ equals $G(\theta, K)$, which is negative by the induction hypothesis. Along the interval $[\theta_K, a_{K+1}], G(\theta, K+1)$ is concave. By an argument identical to the one supporting the base case, there is exactly one value in $[\theta_K, a_{K+1}]$ that satisfies $G(\theta, K+1) = 0$.

We have generalized the induction hypothesis to hold for any value of k greater than one, and for any positive value of θ. Hence, for any k greater than 1, there is a unique positive θ such that $G(\theta, k) = 0$. $\diamondsuit$

Theorem 3 assures us that our resource-allocation policy always produces a well-defined allocation. The host cannot make an arbitrary decision to alter the allocation. Furthermore, unless the agent has specific knowledge regarding its competition, it cannot acheive a more favorable allocation by altering its bid function. Therefore, the allocation that results from our allocation process and agents' optimal bids results in a Nash equilibrium allocation — one in which no agent can benefit by changing its actions in isolation of other changes.

5.4 Incentive

It is reasonable to ask whether an agent can do better by submitting an alternative bid function to $g_i(\theta_1)$. We proved that the i-th agent's optimal bidding strategy is described by $g_i(\theta_1)$ under the assumption that other agents do not account for the i-th's actions. In this section, we discuss the possibility that an agent submits a bid function other than the one that we derive.

To confidently submit an alternative bid function, an agent requires knowledge of its competitors' bids. Two methods for acquiring this information would be to collude with other agents or to infer properties of competing responses to the agent's actions over time.

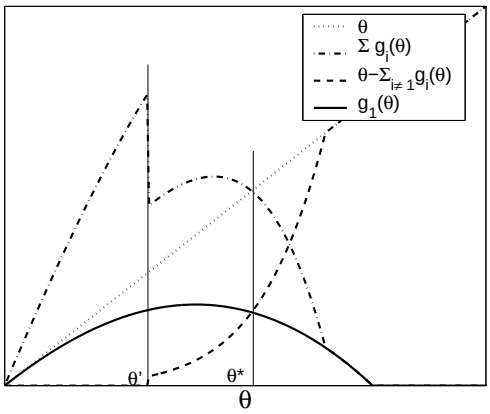

Figure 5.3: A demonstration of an omniscient agent's ability to deviate from the strategy $g_i(\theta)$.

Collusion would allow the possibility that all agents at a site could scale back their bid functions. The host could take steps to discourage collusion by keeping hosted agents' identities anonymous. The anonymity would aggravate an agent's attempts to contact other agents to form a cartel. Once formed, a cartel could easily weaken, since the host only publishes the equilibrium rate of payment, θ, and not each agent's individual bid function. Therefore an agent that deviates from a cartel's collective plan of action could do so secretly, discouraging the formation of such a cartel.

The private nature of agents' bid functions also makes it difficult for an agent to reason about competing agents' responses to a bid. Figure 5.3 shows the situation where the first agent considers discounting its bid in the presence of competition — in this case two other agents. The solid curve represents the agent's proposed bid of $g_1(\theta)$, the dashed-dotted curve represents the population's response, and the upward sloping dashed curve represents the possible allocations that the first agent could receive with perfect information.

The congestion level resulting from the agent bidding $g_i()$ is θ^*. There is a limit on the amount that an agent can discount its bid. A competing agent that is computing its final job has no use for savings in this model. Therefore the competing agent will

always submit a bid that spends its entire endowment, resulting in it paying a rate of α_i/β_i or nothing. The congestion rate corresponding to this event in the example is θ'. Below θ', the competing agent receives all the computation, whereas above θ', the competing agent receives nothing. So by discounting its bid, the first agent risks loss of all computation from an agent with only one job left. Positive gains might be possible, but the agent must have some information concerning the competition's response.

5.5 Simulation

We simulated our allocation policy and compared our policy with three traditional resource-allocation policies to determine the cost of agent prioritization. We show that the cost is small, that our allocation effectively prioritized agents by their endowments, and that our expenditure planning algorithm is insensitive to errors an agent makes in job-size estimation.

We used the Swarm simulation software [LBDL99] to model mobile agents running in a network of 100 hosts. We generated the network topology using the Georgia Tech Internet Topology Model (GT-ITM) [CZ96]. In GT-ITM, a network reflects a hierarchy of sub-networks. Nodes that share a sub-network are likely to be connected, and most traffic to sites outside the sub-network goes up through the hierarchy. Figure 5.4 shows the topology of the network that we used for the simulation. We calculated the delay incurred by an agent migrating between every two hosts at system initialization.

The simulation created agents at a Poisson rate. The agents had exponentially distributed numbers of job to complete. The job sizes were Pareto distributed. The stochastically generated workload reflects bursty resource access commonly observed in computational loads.

We modeled users' preferences that their job complete quickly as a Gaussian

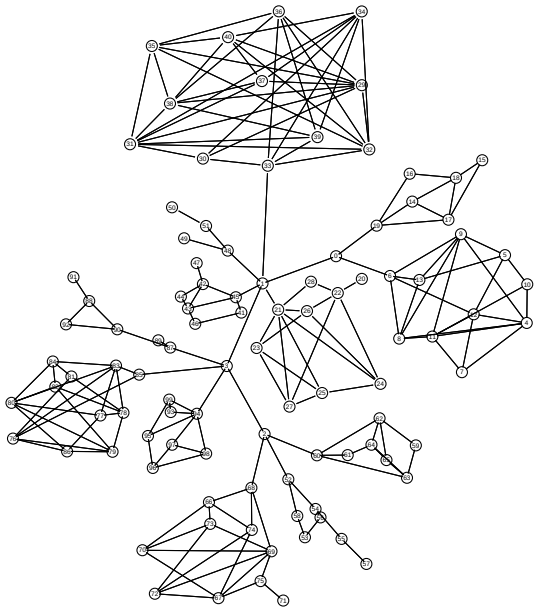

Figure 5.4: The topology of the network used for the simulation.

random variable, truncated so that only positive values were chosen. An instance of the random variable indicates the average amount of currency per computational job unit that the agent may spend to complete its itinerary. We also modeled hosts' processing capacity with a truncated Gaussian random variable.

To measure the qualities of our allocation policy, we implemented similar simulations that used equally-shared, first-come-first-served (FCFS), and shortest-remaining-processing-time (SRPT) allocation policies. The equally-shared allocation policy gives equal amounts of computation to all agents at a site. Each host publishes the number of resident agents and an agent uses these numbers to choose its route.

We implemented a first-come-first-served allocation policy where the earliest arriving agent receives 100% of the available computation. To plan its future expenditures, an agent estimates its prospective host capacity inversely by number of agents that currently visit the host.

The shortest-remaining-processing-time allocation policy preemptively allocates all of a host's computation to the resident agent that would complete its job the

soonest given all agents' current job sizes. For comparison purposes, we assumed that each agent was truthful in its job-size disclosure. To estimate its expected time spent at a prospective host, the agent weights the prospective host's capacity with the agent-population size that the host accommodates.

We used the equal-share allocation because it roughly models the allocation done in deployed mobile-agent systems. FCFS served as a straw-man comparison for how a naive approach might perform. The SRPT is the locally optimal allocation policy; it optimizes the average wait time at a host and provides an upper bound on the performance that can be achieved without coordination between hosts.

To measure the performance of an agent, we divided the agent's execution time by its actual execution time. The ideal time is the result of a shortest-path computation where we computed the shortest time to execute the agent's itinerary in an uncongested network. The resulting performance metric is a real number between zero and one. Zero represents that the agent could not finish its itinerary and one denotes that the agent encountered no resistance.

5.5.1 Route Planning

In this chapter, our focus is the effect of the allocation-policy choice. Therefore, in each of the simulations that we discuss in the chapter, each agent used the same algorithm, Algorithm 2, to choose its next host.

Algorithm 2 chooses the next host to be the one that will complete the next job in the shortest amount of time. It relies on computing the expected computation and transfer latencies at each host. For the equally-shared, SRPT, and FCFS allocation methods, the estimated computation time is

$$\tau_{est} := \frac{N_{ij} q_{ij}}{c_j} \tag{5.38}$$

and we assumed that the agent knows the network latency incurred in migrating to

Algorithm 2 Host selection for agent i

```
1: τ_min ← ∞
2: host ← ∅
3: for each next host alternative k do
4:    τ_est ← expected time at k + travel time to k
5:    if τ_est < τ_min then
6:       τ_min ← τ_est
7:       host ← k
8:    end if
9: end for
10: return host
```

each host. For our market-based allocation, we estimate the time to be

$$\tau_{est} := \frac{q_{ij}(\theta_j + u_{est})}{c_j u_{est}} \tag{5.39}$$

where u_{est} is the value of u_{ij} computed in Equation 5.10 using the last reported value of θ_j for θ_j^{-i} in the equation.

The budgeting algorithm implicitly defined by Equation 5.17 assumes that the agent has already chosen a host for each job in its itinerary. Algorithm 2 commits to choosing a host for a job only when it the agent has finished the previous jobs. In order to budget future expenses, we used the current means of θ_j and c_j for all hosts capable of computing the j-th job. We could have instead used the expected minimum of θ_j and maximum of c_j, but chose a simple heuristic.

5.5.2 Prioritization

We first verified that agents with more currency compute faster than agents with less. Figure 5.5 shows the results of an experiment that examined the effect of endowment on agents' performance. We plot average performance of all agents that ran under SRPT, FCFS, and the shared policies as horizontal lines — endowment was not a variable for those experiments. We group agents under our policy, denoted in the plots as GT data, by their endowments relative to their sums of job-sizes. We weighted

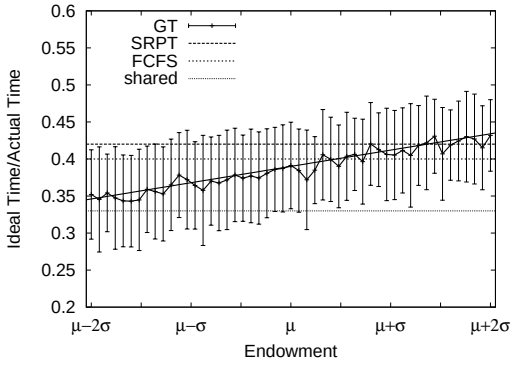

Figure 5.5: Endowment normalized by job-size sum versus performance in a system running at 70% capacity. We plot the mean and standard deviation performance for agents under each wealth level. The SRPT, FCFS, and equally shared polices are plotted as horizontal lines because endowment was not a contribution to the allocation.

an agent's wealth by its computational obligation as a metric for priority. We plot mean and standard deviation performance of agents with endowments two standard deviations, σ, around the mean endowment, μ. We also fit a line to the means to show the regularity of the effect of wealth on performance.

The figure shows the results of experiments that drove the system to 70% utilization. There is clearly a positive relationship between endowment and performance. Note that the mean performance of our budgeting and allocation policies across all endowments was slightly slower, 2% , than the performance under first-come-first-serve. Our policies' performance was about 8% from optimal, SRPT, but 18% faster than shared allocation.

The relationship between endowment and performance strengthened as the system load increased. Figure 5.6 plots endowment versus agent performance. This time, however, agent requests exceeded system capacity by 40%. Our allocation policy was able to discriminate and ignore poorer agents. Under the other allocation policies, an overloaded system never reaches equilibrium, and we cannot measure average agent performance.

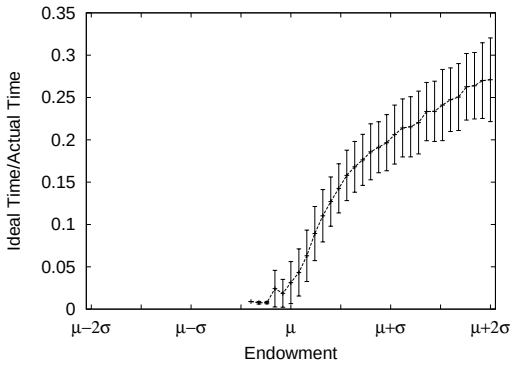

Figure 5.6: Endowment normalized by job-size sum versus performance in an over-constrained system.

What to do with the poorer agents is an implementation decision. The host machine may refuse to take an agent that does not have a threshold endowment, or it may store the agent for later execution when congestion subsides.

5.5.3 Network Delay

The next two sections look at the importance of accurate information to guide an agent's decisions. Our first such experiment compared the performance of agents running under the four allocation policies when the information that agents use to plan their budgets and itineraries was aged. The aging of data simulated the effect of lag incurred by data transferred over the network.

Figure 5.7 plots the average performance of agents operating under different network delays. The performance of all four systems degraded in a similar fashion as network delay increased. Figure 5.8 shows that the relationship between endowment and performance held as network delay increased.

5.5.4 Estimation Error

In an application, it is doubtful that an agent will have perfect knowledge of its job requirements. Our budgeting algorithm requires an agent to know q_{ij}, the size of

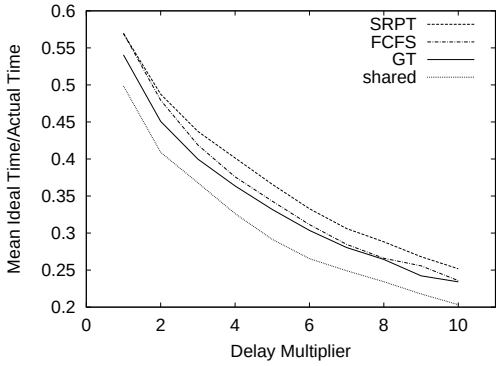

Figure 5.7: Network delay and its mean effect on agents' performance.

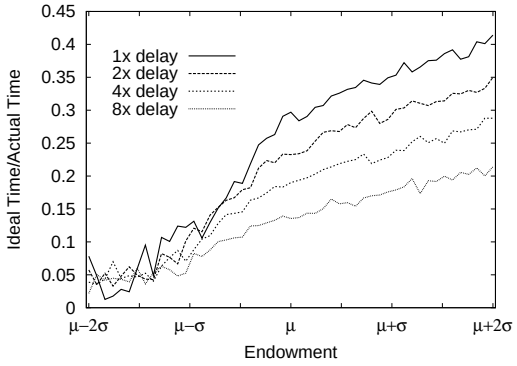

Figure 5.8: The effect of endowment on agents' performance at different levels of network delay. We omit error bars to present a neater plot.

each job it executes, however. We investigated the effect of errors in q_{ij} on an agent's performance.

We modeled the actual job size as a Gaussian random variable with mean 1 multiplied by the agent's estimate. To construct a bid function, each agent biased its actual job size by the amount of the estimation error for that job. The bias for each job was determined when the simulation created the agent. Figure 5.9 shows how performance of the SRPT and our market-based policies deteriorated as a function of the job-size estimation error. It turns out that performance dropped only 3% for every multiple of the mean by which we increased the deviation. Each agent's performance was relatively insensitive errors in its job-size estimates.

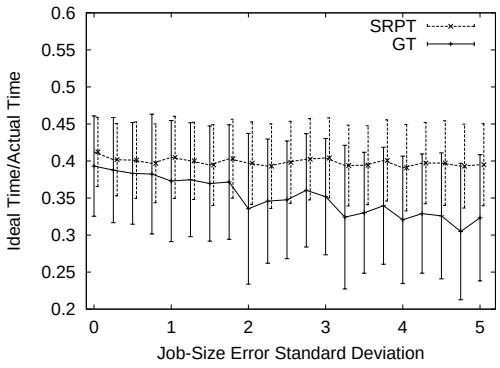

Figure 5.9: The effect of job-size estimation error on agents' performance.

5.6 Conclusions

In this chapter we present a mechanism that determines relative agent priority through a single parameter: an agent's monetary endowment. The endowment expresses the user's preference that the agent completes quickly as well as the agent's ability to inflict congestion upon other agents. The framework that we present in this chapter is appropriate when the system's users collectively own the computing resources. We designed the allocation mechanism with the guideline that increased system utilization yields higher performance and that an agent's performance should be expressed through a simple parameter.

We reason about an agent's decisions about how to spend to optimize its performance. The result is an optimal budget-planning algorithm, given that the agent has rough estimates of its hosts' congestions and capacities and its own job size, but no other knowledge that concerns the state of the network. The algorithm is no more sensitive to network delay than other allocation polices that we consider and is robust to errors in the agent's job-size estimation.

We measure the cost of prioritization through comparison with the optimal policy that minimizes the average agent wait time at a host. The cost of our prioritization is about 8% of the optimal performance, but our prioritization allows the system to

run important jobs when the system is over-taxed.

The implementation from theory to simulation is simple. We compute an agent's optimal performance analytically, and a host needs only to collect the parameters that describe each agent's bidding behavior and run a simple one dimensional search. Implementation in a real system is relatively straightforward; Cooper and Gray [CG00] implemented our allocation and planning algorithms inside a version of D'Agents that runs on top of QLinux, a version of Linux that supports proportional CPU scheduling.

α_i	A parameter that describes the i-th agent's bid function. $\alpha_i := E_i - \sum_{j \neq 1}^{M_i} q_{ij} \theta_j^{-i} / c_{ij}$
β_i	A parameter that describes the i-th agent's bid function. $\beta_i := q_{i1} / c_{i1}$
γ_i	A parameter that describes the i-th agent's bid function. $\gamma_i := \max\{\epsilon, \sum_{j \neq 1}^{M_i} q_{ij} \sqrt{\theta_j^{-i}} / c_{ij}\}$
$f_{ij}()$	The i-th agent's bid given all other agents' bids, defined by Equation 5.10. The rate at which the i-th agent pays the host conditioned on the rate at which the host collects from all agents excluding the i-th.
$g_{ij}()$	The i-th agent's bidding function at the j-th host, defined by Equation 5.17. The rate at which the i-th agent pays the host conditioned on the rate at which the host collects from all agents.
τ_{ij}	The time the i-th agent spends at the j-th host.
θ_j	The sum of all bids at the j-th host.
θ_j^{-i}	The sum of all bids, except the i-th agent's, at the j-th host.
U_i	The i-th agent's utility.
u_{ij}	The rate at which the i-th agent pays the j-th host.

Table 5.1: Notation introduced in Chapter 5.

Chapter 6

Host Revenue Maximization

The model we define in Chapter 5 has many desirable properties, but it is based upon several assumptions we relax in this section. The utilities of resource owners and users may not be realistic. Specifically, in Chapter 5, resource owners have no value for the revenue they collect nor the computation they allocate to agents. A more realistic model would allow hosts to take a more active role in resource allocation. A host would attempt to maximize revenue generated by visiting agents; thus resource owners have incentive to accommodate arbitrary agents, not just ones belonging to a common organization.

From the user's perspective, the model from Chapter 5 has no established link between utility and the precious endowment that the user imbues to an agent. The model neglects that a user might place proportionally higher values on elimination of longer latencies than shorter ones [WS98, EWCS96]. Additionally, a user's utility frequently depends on expectations and previous experiences [WS98].

In this chapter, we expand and modify the model presented in Section 5 to ground the value of the currency used to purchase computation and to provide a more realistic model of users' utility. We begin in Section 6.1 by summarizing relevant results from game theory concerning utility and mechanism-design theories. We determine that ideal resource allocation is infeasible and relax our objectives to arrive at a more

pragmatic view.

In Section 6.2, we describe several different families of utility functions commonly used in microeconomics. We are particularly interested in the model in Section 6.2.3, where we model an agent's utility as the weighted sum of the agent's ability to complete in time less than its user's expectation and the savings the agent is able to return to the user. Section 6.2.4 describes existing work that defines a mechanism that maximizes expected revenue with few assumptions about agents' utilities. We arrive at an allocation mechanism that we describe in Section 6.3 and define the algorithm to calculate the allocation in Section 6.4.

We derive two bidding strategies for use with the mechanism and the expectation-based utility model in Sections 6.5 and 6.6. The first strategy is less practical and is useful for motivating the second, but its use illuminates the value of hosts approximation in the allocation mechanism as a means to motivate agents to bid more competitively. Through approximation of the optimal revenue-maximizing solution, an agent perceives a more accurate picture of the nature of competition among agents than if the host computes the allocation exactly. The second, on-line, strategy uses a heuristic to compress the space of useful bids and Bayesian techniques to build a belief function to express the utility derived from simple bidding behavior. We simulate agents using this strategy and present the results in Section 6.7. We find that agents under the new model behave in many of the same ways that agents in Chapter 5 do, but that an agent has less incentive to identify its priority to the host. The result is that some low priority agents are served and that the allocation is not as efficient as a situation where agents divulge more information.

6.1 Objectives

There are several shortcomings in the model presented in Chapter 5 stemming from agents' utility functions and host motivations. Hosts are motivated to altruistically

assist agents by striving for high resource utilization and agents have no incentive to return their remaining endowment to their users. It is reasonable to question both of these assumptions. If we relax them, we will be able to synthesize a market environment that is usable in open environments with self-interested hosts and agents. To this aim, we investigate the impact of utility maximization on mechanism design. We find that game-theoretic ideals are likely unobtainable. We update our objectives in that light and propose a new mechanism for computation allocation. We propose two realistic utility functions useful for modeling different application types and analyze how agents and hosts use the new mechanism. We conclude the chapter by testing the model through simulation.

The principal, the host, presents agents with a mechanism for negotiating allocation of and payment for resources. In the scenario we present in this Chapter, the chief motivation is host utility maximization. The mechanism designer often attempts to achieve other goals that facilitate agents' strategy construction. An ideal mechanism would be one that is individually rational, incentive compatible, and has a dominant strategy for all agents.

A rational agent attempts to maximize its utility given its beliefs about the world. For the work in this dissertation, we assume that agents always have a choice to participate in a mechanism or not. If the agent chooses not to participate, it receives fixed level of utility called its *reservation utility.* A requirement for a rational agent to use a mechanism is that the mechanism must be *individually rational;* the expected utility of participating in the mechanism must be at least as much as the agent's reservation utility [FT96, p. 246].

A principal that optimizes a mechanism has to acknowledge agents' preferences. Normally, preferences are private information, but there exist mechanisms where an agent's *dominant strategy* is to reveal preferences that describe its type. A dominant strategy is one that optimizes the agent's strategy irrespective of the other agents'

actions [OR96, p. 18]. A *Nash equilibrium* is one where each agent acts to maximize its utility, through its best response, and no agent can improve its state through unilateral action. When the dominant strategy for a mechanism is for an agent to truthfully reveal its preferences, we say that the mechanism is *incentive compatible,* or *truthfully implementable.*

The canonical example of an incentive compatible mechanism is the second-price sealed-bid auction [Vic61]. The principal wishes to sell an object, but has no information of the object's value. Each player of the game, however, knows exactly what the value of good is to herself. If all players have private valuation of the good (that is, given that a player loses, she is indifferent which one of her competitors receives the good), then the mechanism presented in Algorithm 3 is an incentive-compatible dominant-strategy mechanism to sell the good.

Algorithm 3 Second-price sealed-bid auction (Vickrey 1961)

1: All players simultaneously submit bids to the principal.
2: The winning player is the player with the highest bid.
3: The winning player pays the principal the amount of the highest *losing* bid.

The optimal strategy for each player is to submit a bid equal to its valuation for the object.

The strategy assumes that a player loses nothing by revealing her valuation to the principal and other agents. The cost of revelation becomes important in scenarios with repeated interaction, however. If the principal sells more than one object, it can learn the players' valuations and the sealed-bid second-price auction does not necessarily maximize revenue given knowledge of the value of the good [FT96, p. 253]. The gap between bidders' utilities may be large enough to justify the principal to place a substantial reserve price, a price under which no sale occurs.

Incentive compatibility is an attractive goal because it leads to accurate communication of agents' preferences. A host's uncertainty of agents' preferences leads to

efficiency losses in the allocation computation. Myerson and Satterthwaite state that it is not possible for traders to exhaust surplus from trade so long as there is a positive probability for gains from trade, a positive probability of no gain from trade, and the traders have incomplete information [MS83].

Not all games have dominant strategies, however, and even fewer are truthfully implementable. The Gibbard-Satterthwaite theorem states that a mechanism that is truthfully implementable in dominant strategies if and only if the mechanism is *dictatorial* — one where a single agent determines the outcome — and there are no restrictions on agent's utilities and at least three agents participate [Gib73, Sat75].

The Gibbard-Satterthwaite theorem leaves room for hope. We may assume agents' utilities fall into a restricted domain of preference relations. The second-price auction described in Algorithm 3 avoids the fate of the Gibbard-Satterthwaite theorem under the assumption that each player has a private valuation for the good. An agent is indifferent to all the outcomes in which it does not receive the good.

Alternatively, we can restrict the set of outcomes. There are situations where there are truthfully-implementable dominant-strategy mechanisms for allocation of public goods. Here we restrict the set of outcomes to two events: undergo a project or forgo it. Such mechanisms are called Groves mechanisms. We shall see, however, that there is often substantial cost to the principal in implementing such a mechanism. Groves mechanisms are used to decide whether or not to produce a public good [Cla71, Gro73]. Let the public good have cost v_0 and v_i is the i-th player's value of the good. Furthermore, the i-th player pays an amount $h_i(a_{-i})$ regardless of whether the public good is produced. The amount $h_i(a_{-i})$ can be any amount so long as it is independent of a_i, although, it may depend on a_{-i}, the actions taken by the other players. Algorithm 4 has an incentive-compatible dominant strategy for all players. The dominant strategy for every player is to reveal her valuation for the public good because the payment that a player makes is independent of her bid.

Algorithm 4 The Groves mechanism (Clarke 1971, Groves 1973)

1: All players simultaneously submit bids, a_i, to the principal.
2: **if** $\sum_i a_i > v_0$ **then**
3: Produce the good.
4: Each player i pays $h_i(a_{-i}) + v_0 - \sum_{j \neq i} a_j$.
5: **else**
6: Do not produce the good.
7: Each player i pays $h_i(a_{-i})$.
8: **end if**

While the Groves mechanism has a dominant strategy that every player reveals her valuation, it does have a serious drawback. The principal does not always collect payment greater than the cost of the good. The constraint that the principal must collect at least as much revenue as it pays out is called *budget balance.*

Without restrictions on agent utilities, the Groves mechanism is the only truthfully implementable dominant strategy mechanism. It is also the case that, in general, it is not possible for the Groves mechanism to satisfy budget balance [GL77].

Both the Groves mechanism and the second-price auction demonstrate that there are often substantial costs involved in ensuring incentive compatibility and dominant strategies. Given that incentive compatibility can be costly and that few mechanisms yield dominant strategies without constraining assumptions, we relax our objective to find a mechanism that maximizes a host's utility given its beliefs. We also assume that each agent maximizes its utility under its beliefs. Thus, our goal becomes to devise mechanisms that yield easily computable equilibrium.

A *Bayesian-Nash equilibrium,* is an outcome where each agent follows its best response given its beliefs about the world [OR96, p. 26]. Typically, Bayesian-Nash equilibrium are difficult to compute, but we can often compute an agent's actions by placing simplifying assumptions on an agent's knowledge.

6.2 Utility

It is now useful to look at examples of utility functions before we consider mechanism design any further. We examine several utility models as well as performance metrics on which utility could be based in this section.

Utility is a measure of an agent's satisfaction with the outcome of its and others' actions. A rational agent attempts to maximize its utility given its understanding of the mechanism in which it participates. We express utility as a numerical value that we use to compare the agent's preferences of different outcomes. In general, it is not possible to directly compare two agents' utilities; there are no meaningful units to compare the utility of one agent with that of another.

6.2.1 Cobb-Douglas Utility

The *Cobb-Douglas utility* function is a commonly used utility function in introductory microeconomics [MCWG95, p. 55]. It is expressed as the sum of the logarithms of the amount of each good, x_i, consumed in the set of available goods, P, as

$$U = \sum_{i \in P} \alpha_i \log x_i \tag{6.1}$$

or alternatively as

$$U = \prod_{i \in P} \alpha_i^{x_i}, \tag{6.2}$$

where $1 = \sum_{i \in P} \alpha_i$ and α_i denotes the relative importance of the ith good.

The Cobb-Douglas utility function has two important features: diminishing returns and the wealth effect. As an agent increases its consumption of a good, the amount of additional utility derived from consumption decreases. In effect, the relative marginal value of other goods increases as the agent consumes more of one good. Intuitively, this makes sense: drinking ten beers does not make one ten times as happy

as drinking just one; and consumption of those ten beers increases one's relative value for other goods, like aspirin.

If one of the goods to be consumed is savings, or future expenditure, then an increasing (decreasing) an agent's endowment would result in higher (lower) consumption of other goods. This effect is called the *wealth effect* [MCWG95, p. 24]. The wealth effect is the relationship between an agent's consumption of a good and its wealth.

In computation markets, the wealth effect is useful. It allows a user to express the priority of a job through a single parameter — the endowment. As the agent's endowment increases, so will the rate at which it consumes resources. Weighting the relative values of computational priority and endowment can be tricky, however, especially in the face of uncertainty of future consumption.

6.2.2 Quasi-Linear Utility

Reasoning about utilities is often slippery. In many cases, it is impossible to compare the utilities of two different agents. If the agents' utility functions translate to a common domain, however, then utilities of different agents can be compared. The common denominator for quasi-linear utility functions is money. A *quasi-linear utility* function is a function of the amounts of goods consumed, $f(X)$, minus the monetary cost of consumption, $t(X)$ [MCWG95].

$$U = f(X) - t(X) \tag{6.3}$$

Here, every level of utility corresponds to a set of pairs of consumption and expenditure. Because quasi-linear utility can be directly solved for a price that the agent would pay to achieve any utility level, the utilities of agents using quasi-linear utility functions are comparable.

The drawback with quasi-linear utility functions is that they exhibit no wealth

effect. In optimizing its utility given a budget constraint, an agent consumes until the marginal value of consumption is no longer greater than the marginal value of savings. Mathematically, this decision stems from taking the partial derivative of the utility function with respect to endowment. The result of the calculation only relies on the agent's endowment as a boundary condition.

In a computational-resource allocation scenario, we would like to see processes with higher priority consume faster. We express a process' priority through its endowment, but a rational agent using a quasi-linear utility function will not change its actions if its endowment changes, with the exception of constraints. If we are to use quasi-linear utility, we must base an agent's utility on a metric other than throughput.

6.2.3 Metrics: Other Notions of Utility

Utility is based upon a metric. An agent has a valuation for holding or consuming an amount of a good upon which we can base a utility model from Section 6.2.1 or 6.2.2. For example, an agent may be only concerned with getting its job done as quickly as possible given the available resources. In this case, we can drop the savings from an agent's goals as we did in Chapter 5. The reasoning led us to the straightforward optimization of the total completion time, or end-to-end latency, which is equivalent to optimization of throughput in our case. We now explore other applications of metric-based utility functions: meeting expectations and maximization of response time.

Expectation-Based Utility

At the end of the day, utility has to have a meaningful connotation with both a user's level content as well as the agent's actions. Inferring a user's preferences may be difficult [WS98]. Users may have different expectations and experiences that affect

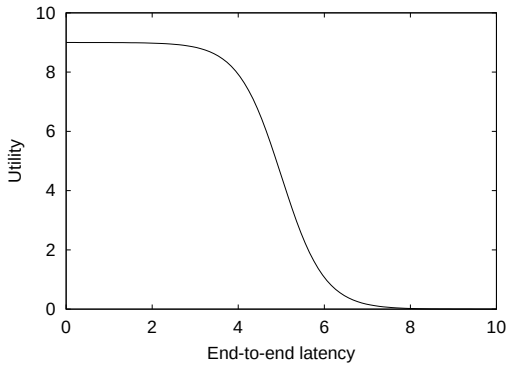

Figure 6.1: An example sigmoidal utility function.

their perceptions of latency.

We assume that a user has a notion of the time that the i-th agent will take to execute its itinerary. We define this latency as $\hat{\tau}_i$. Furthermore, the user is willing to spend an amount E_i, that the user gives to the agent to complete the jobs. The user is content if the agent finishes in time limited by roughly $\hat{\tau}_i$ and the agent returns with remaining currency. We combine the idea of this expectation and quasi-linear utility to define the agent's utility as

$$U_i = \frac{E_i}{1 + \exp\left(\kappa_i \left(\sum_{j=1}^{M_i} \frac{q_{ij}}{c_{ij} x_{ij}} - \hat{\tau}_i\right)\right)} - \sum_{j=1}^{M_i} t_{ij}, \tag{6.4}$$

where $\kappa_i > 0$ is a parameter that describes the precision of the user's expectations and, again, $q_{ij}/c_{ij}x_{ij}$ is the latency incurred in executing the j-th job and t_{ij} is the expense involved in executing the j-th job. Figure 6.1 illustrates an example of an expectation-based utility function holding the cost of execution fixed. We assume that the agent has the option of not participating with the result of zero utility — the reservation utility. The threshold motivates an agent's decisions on if, when, where, and how to execute its jobs; if the expected utility of the itinerary is less than zero, then the agent fails and returns its remaining endowment to the user.

To save space, in the context of our expectation-based utility, we define

$$h(x_{i1}) := \exp\left(\kappa_i\left(\sum_{j=1}^{M_i}\frac{q_{ij}}{c_{ij}x_{ij}} - \hat{\tau}_i\right)\right). \tag{6.5}$$

Response-Time Based Utility

The notion of utility corresponding to end-to-end latency captures jobs that do not require user interaction such as compiling programs, information retrieval, and image processing. Much of the motivation for using mobile agents is to minimize latency, but often an agent's latency is perceived prior to the agent finishing its jobs. For example, an agent may update the user on its progress, present media whose value declines with delay, or require repeated interaction with the user. We therefore present another metric of utility to correspond to response time:

$$U_i = E_i - \sum_{j=1}^{M_i}\left(\frac{q_{ij}}{c_{ij}x_{ij}}\right)^2 - \sum_{j=1}^{M_i} t_{ij}. \tag{6.6}$$

Here, E_i is the agent's endowment as well as an upper bound for the utility that the application could deliver given no event-oriented latency or expenditure. Again, we assume that the user has a reservation utility equal to zero. We weight longer delays more heavily than shorter ones by squaring the latencies, $q_{ij}/c_{ij}x_{ij}$. The choice to raise the latency to the power of two is arbitrary and the same analysis would hold for any exponent greater than one. The larger the exponent, the more heavily larger latencies are weighted.

6.2.4 Host Utility

One of the objectives for mechanism design in this section is to maximize the host's utility. We assume that the host's utility is formulated as a linear combination of the amount of resources unsold and the income generated by the sold revenue. A host's utility function is

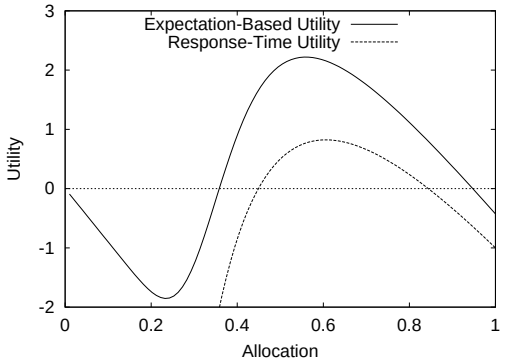

Figure 6.2: Examples of expectation-based and response-time utility derived from executing at a host with a fixed price of computation.

$$U_0 = v_0 \left(1 - \sum_{i=1}^{N} x_{i1}\right) + \sum_{i=1}^{I} t_{i1}. \tag{6.7}$$

The parameter v_0 is the host's private value of the usage of its computation. We again use the shorthand $t_{ij} = t(x_{ij})$, where $t(x_{ij})$ is the amount the i-th agent pays for its j-th resource as a function of the resource quantity that the agent receives. Our goal as the mechanism designer is to present a mechanism that allows hosts to maximize U_0 under the constraint that no more resources are sold than held:

$$\begin{aligned} &\max_{x_{ij}} v_0 \left(1 - \sum_{i=1}^{N} x_{i1}\right) + \sum_{i=1}^{N} t_{i1} \\ &\text{s.t. } 1 \geq \sum_{i=1}^{N} x_{i1}. \end{aligned} \tag{6.8}$$

6.3 Mechanism

In this section, we motivate the choice of our resource-allocation mechanism — a repeated first-price sealed-bid auction. We describe complications in price calculation that a host faces. Our conclusion is for a host to use a simple, analyzable mechanism. A host auctions time slices of access to agents. Agents submit bids as tuples that describe the quantity or priority of computation that they wish and the price the

agent is willing to pay for computation and the host chooses the set of tuples that maximize its revenue for the time slice.

6.3.1 Motivation

We begin by examining decisions that optimize a host's revenue. Maximization of Equation 6.8 is difficult. The function $t(x_{ij})$ is information private to the i-th agent and hence unknown to the host.

In real-world economies, sellers face a similar problem everyday. Fluctuating demand or insufficient information to calculate a price motivate a seller to use an auction. Our computational environment is dynamic. Computational workloads are, in general, bursty in nature.

The broad nature of the uses for computation make it difficult to establish an efficient price for computation. This last feature is aggravated by the fact that there may be an unknown number of competing vendors. We quickly summarize the properties of a market when there are many sellers, few sellers, and exactly one seller.

Computation sold by one host may be substitutable for computation at another. In the worst case for the host, the agent may have no preference other than the rate at which it computes. Economists call this scenario perfect competition when the number of substitutable hosts is large and no single buyer's actions have a substantial effect on the market. Under perfect competition, the revenue-maximizing fixed price of an item is equal to its cost (which includes the opportunity cost of capital).

A better case for the host would be to limit the number of competitors. For example the host might be one of few sites that is located closer to frequently updated data sources. The agent may only have the choice of executing at a handful of hosts and have weak preferences among the choices. Economists call a market with few sellers and many buyers an oligopoly. Price computation in oligopoly is a complex process that heavily relies on assumptions of agents' values for the good to be sold.

The best situation for a seller is to face zero competition. In a computation market, the host may have the only database that handles the agent's desired information or the host may be situated in the network such that it is the only desirable host on which to operate. These examples describe instances of monopolies — cases where there is a single seller of a good. Under monopoly conditions, revenue-maximizing prices are easy to compute in comparison to the oligopoly scenario.

The conditions that determine the substitutability of a good are private information to the agent. Different agents competing for the same resource may have different abilities to substitute the resource. Thus several agents that compete for computation at a site may effectively be in different market types. One agent might interface with a local database, another may perform generic computation, and another wish fast network access to its user or to an Internet auction.

6.3.2 Solution

Given the uncertainty in agents' utilities and competition, we investigate agent behavior under a simple mechanism. Our motive is to present a mechanism that both agents and researchers can analyze. Fortunately, the simplification is not costly to the host. The revenue-equivalence theorem states that for any two auctions that yield the same probability of allocation to a buyer, also yield identical expected revenues to the seller so long as buyers' preferences are statistically independent [MCWG95, p. 890].

In the pursuit of simplicity and in light of the revenue cost, we examine the situation where the host uses a sealed-bid auction to allocate computation. The host has a secret per-unit private value of computation, v_0, and computes a reserve price, $v*$. The host announces that it will sell computation for Δ time units and that there is a secret reserve price. Each agent submits a bid (x_i, t_i) that states that the agent will pay t_i currency for the portion of the computation equal to x_i over the next

time slice of Δ seconds. The host ignores all bids i such that $t_i/x_i < v*$ and of the remaining bids picks the set that maximizes revenue under the restriction that the resulting allocation is feasible.

Reserve-Price Computation

The calculation of the host's reserve price is an important issue if the host is to maximize its revenues in repeated auctions. Obviously, it is not desirable to the host to sell computation at a price lower than its value. The host's private value for computation may not reflect fixed costs as it is not usually possible to store generic computation for future use. The value of computation to the host would represent the electricity required to power the processor and supporting devices as well as the ability to perform private computation.

The host may want to increase its reserve price, $v*$, above its private value, v_0, to foster competition among the agents and increase its revenue. At most, a host would increase its reserve price to the monopoly price of computation that satisfies the equation

$$v* - \frac{1 - P(v*)}{p(v*)} = v_0. \tag{6.9}$$

where $P(v)$ and $p(v)$ are, respectively, the cumulative and probability density functions for an agent having marginal value of computation of v [FT96, p. 285].

A lower bound for the reserve price is the cost of computation. This price is the market equilibrium price under perfectly competitive scenarios, the environment that we simulate in Section 6.7. Note that the cost of computation is not simply the immediate cost of the host forgoing calculation, but includes the expected return of capital in a competitive market. For example, a principal may expect a 5% return from an investment of comparable risk to computation sales and would then set the price of competitive computation to be 5% greater than cost.

6.4 Allocation Computation

We now describe the process of allocation computation. The computation solves an NP-hard problem, so we consider algorithms that approximate solutions.

The problem can be stated as follows:

$$\begin{array}{ll} & \max \sum_i t_i x_i y_i \\ \text{s.t.} & \sum_i x_i y_i \leq 1 \\ & y_i \in \{1, 0\} \end{array} \tag{6.10}$$

y_i are indicator variables, x_i denotes the agent's requested allocation, and t_i is the i-th agent's payment for winning the auction. Determining the set of bids that maximizes revenue under the resource availability constraint is an instance of the knapsack problem [PS98].

We use an existing dynamic-programming algorithm [Hoc97b, PS98] to maximize the objective function first long ago [IK75, Law77]. The algorithm works by partitioning bids into sets of "big" bids and "small" bids. We optimally choose "big" bids that maximize revenues, given the quantity available to be sold. In the event that there is remaining quantity for sale, we greedily sell the remaining computation using the price-per unit as our guiding heuristic. The algorithm runs in reasonable time because it is possible to bound the number of "big" bids as a function of the degree of approximation of optimality.

The algorithm first orders the bids in non-increasing order of t_i/x_i. Let

$$t_{\max} := \max_{i \in [1 \ldots N_{ij}]} \{t_i\}, \tag{6.11}$$

$$\bar{i} := \max_i \left\{ \sum_{k=1}^{i} x_k \leq 1 \right\}, \tag{6.12}$$

$$t_0 := \max \left\{ t_{max}, \sum_{i=1}^{\bar{i}} t_i \right\}, \tag{6.13}$$

where, again, N_{ij} is the number of agents (bids) at the j-th host. A "big" bid is a bid (x_i, t_i) such that

$$t_i \geq \frac{1}{2\epsilon t_0}, \tag{6.14}$$

where ϵ, here, is the gap by which the host wishes to approximate the optimal selection of bids.

The number of big bids is bounded by $O(1/\epsilon^2)$. The host optimally packs the big bids through dynamic programming with scaled values that allow the desired approximation. The remaining bids, the small ones, are sorted by density, t_i/x_i, and packed greedily. The algorithm runs in time

$$O\left(\frac{1}{\epsilon^4} - N_{ij}\log_2\epsilon\right) \tag{6.15}$$

where

$$\epsilon := \frac{T_0^2 + 1}{2\epsilon T_0^2} \tag{6.16}$$

and s is the scaling factor that the host uses for the packing,

$$s := \frac{1}{4\epsilon^2 T_0}. \tag{6.17}$$

6.5 Fictitious Play

We now present how a myopic agent can attempt to maximize its expected utility through learning over iterated rounds of simulated bidding, or fictitious play [FL98]. Fictitious play is a technique used to learn how an agent should participate in a sequence of auctions when an agent's utility from the outcome of one auction depends on another auction in the sequence [BGS99]. The agent uses the outcomes of its actions in previous rounds of play, in our case auction participation, to learn a strategy

for play in future games. We are interested in the approach because of the limited assumptions on an agent's intelligence. We will show that the host can guide agents to bid more competitively through approximating the optimal bid choice.

6.5.1 Naive Search

We begin by examining the performance of simple algorithm. An agent has a utility function $U_i(x_i, t_i)$ that is the expected utility of repeatedly bidding (x_i, t_i). The agent knows its utility from an allocation, but there is uncertainty in the outcome of any particular bid. The agent does not know why a particular bid is rejected. The reason could be that the bid price is beneath the host's reserve price, or that there is a more attractive competing bid from another agent.

The agent's only knowledge comes from its experience bidding. It attempts to maximize its expectation-based utility through a gradient-ascent search of its utility function. Algorithm 5 sketches the operation of the agent's gradient-ascent search that we use with fictitious play. The parameters of the search are number of repetitions an agent performs before changing its bid, the precision of the approximation that the host uses to pack the bids, and the step size that the agent uses to update its bid.

Figures 6.3– 6.5 show how an agent's behavior changes over time in simulations of four experiments. The simulations had two variables: the approximation used by the host in bid packing (50% versus 90% approximation) and the number of times the agent repeats its bid before updating.

Figure 6.3 shows how agents learn over time to utilize the host's resources more effectively. Agents that repeat their bids initially learn more slowly, but they eventually learn better allocations. The host can also guide agents' behavior by approximating the packing. The approximation yields a softer, easier to perceive gradient along which the agents can search, especially when agents repeat bids.

Algorithm 5 Gradient-ascent bidding algorithm

variable uses	
(x_i, t_i)	A simplex with which to calculate the gradient.
v_i	Values on the expected utility on the simplex.
∇v	The change in expected utility on the simplex.

1: Compute feasible bid space
2: $(x_2, t_2) \leftarrow (1.0, t_{max}(1.0))$ {Initialize the gradient with two}
3: $(x_1, t_1) \leftarrow (x_{min}, t_{min}(x_{min}))$ {known points and a random one.}
4: $\hat{v}_0 \leftarrow v_1 \leftarrow v_2 \leftarrow 0$
5: $(x_0, t_0) \leftarrow$ uniform random point in feasible bid space
6: $i \leftarrow 0$
7: **repeat**
8: $v_0 \leftarrow$ mean utility bidding (x_0, t_0) n times
9: $v' \leftarrow$ mean (v_0, v_1, v_2)
10: $\hat{v} \leftarrow \max(\hat{v}, v')$
11: Compute gradient, ∇v, from $\{((x_j, t_j), v_j)\}, j \in [0 \dots 2]$
12: $(x', t') \leftarrow (x_0, t_0)$
13: **if** $\nabla v \neq 0$ **then**
14: $(x_0, t_0) \leftarrow (x', t') + \gamma(1 + \lambda^i)(\hat{v} - v_0)\nabla v$
15: **else if** $v_0 = 0$ **then**
16: $(x_0, t_0) \leftarrow (x', t') + \alpha$
17: **end if**
18: **if** Bounds check (x_0, t_0) from (x', t') fails **then**
19: **goto 5**
20: **end if**
21: $j \leftarrow \arg\max_j(|x_0 - x_j, t_0 - t_j| : \{(x_k, t_k) : k \neq j\}$ is not co-linear)
22: $(x_j, t_j) \leftarrow (x', t')$
23: $i \leftarrow i + 1$
24: **until** market convergence

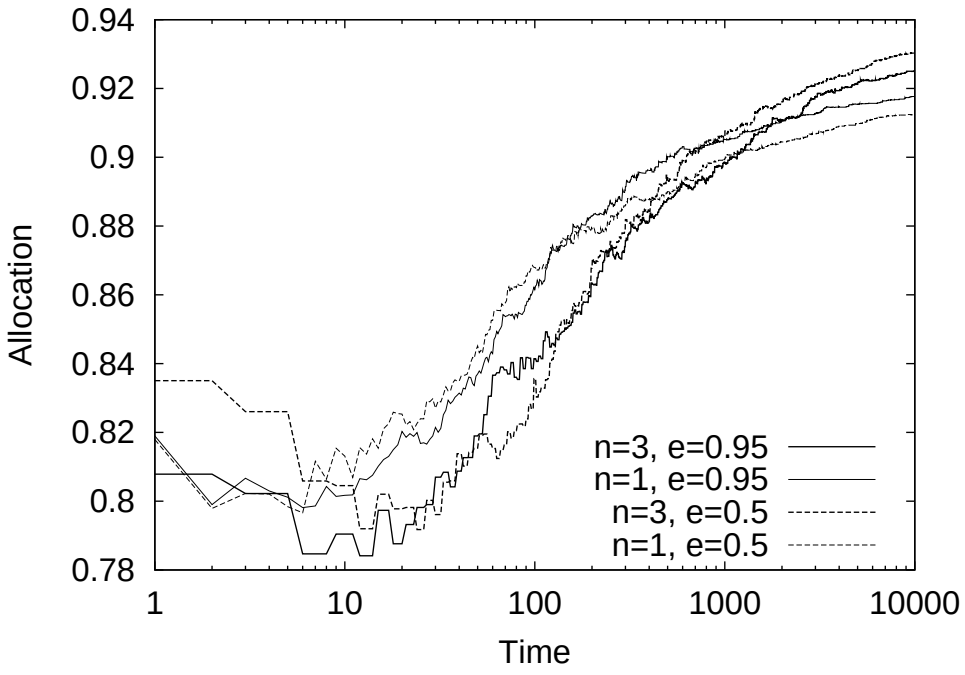

Figure 6.3: The change in system efficiency under different experiment parameters.

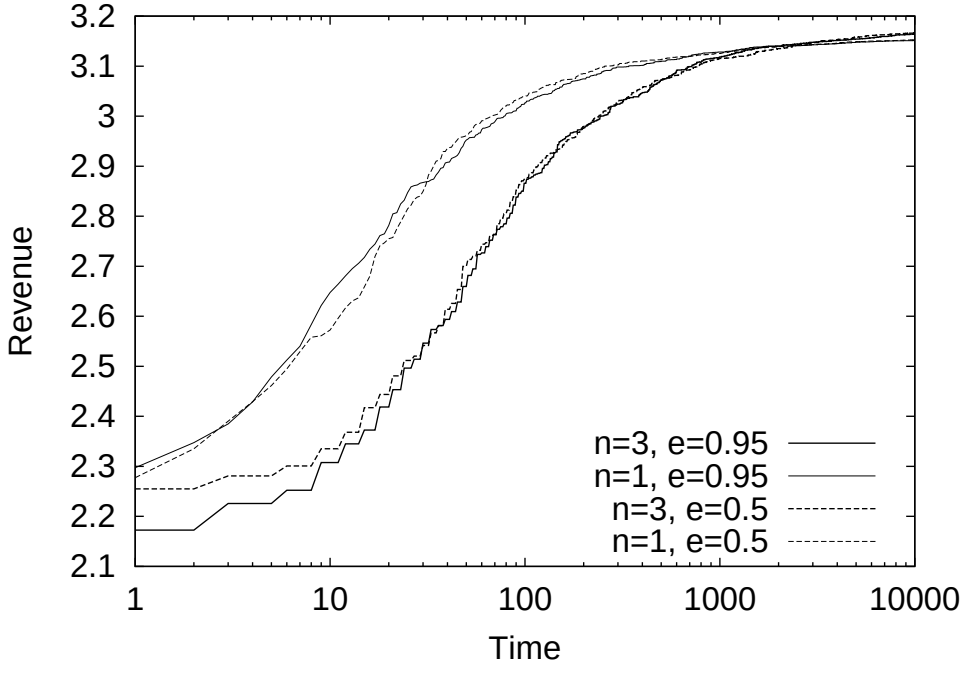

Figure 6.4: The change in host revenue under different experiment parameters.

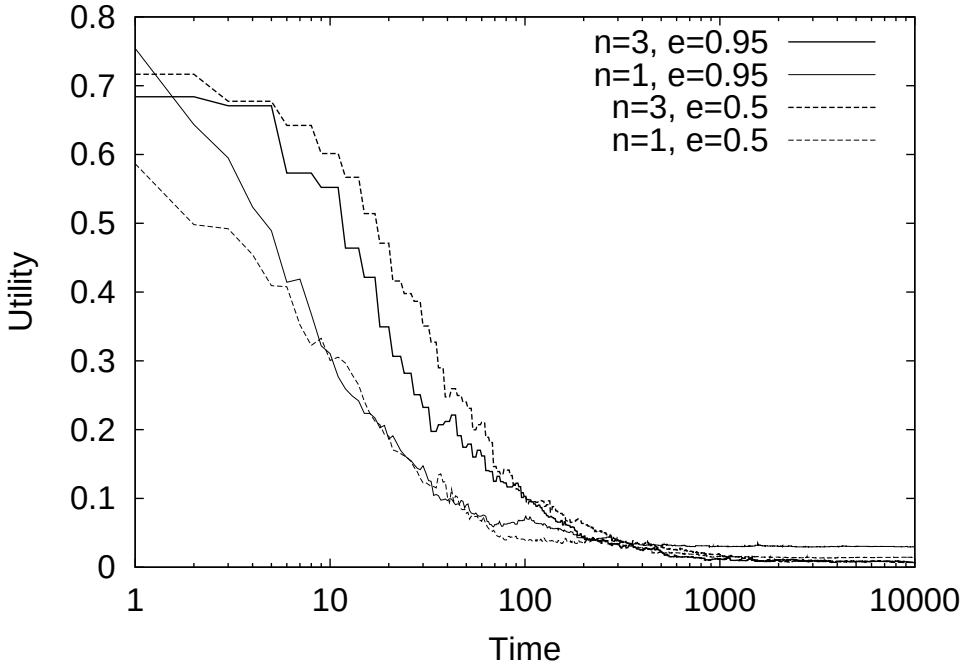

Figure 6.5: The change in an agent's utility under different experiment parameters.

Figure 6.4 shows that neither variable has any real effect on a host's revenues. The revenue differences between the two approximations are not significant. As one might expect, the revenue curve of the repeating bid experiment lags behind the non-repeating experiment by a factor of three before revenue catches up.

Over time, agents become more competitive and their utilities fall off to nothing as we illustrate in Figure 6.5. Like the revenue plot, neither variable had a surprising result.

The gradient-ascent bidding method provides us with assurance that an agent can learn with limited information, but the rate of learning is slow and probably not reasonable for an on-line solution. An agent's perception of the local gradient has errors. If other agents' recent bids are not competitive with the bids of one agent, the agent will perceive a flat spot in the utility landscape. Through the course of learning, the agent's utility changes as competing agents update their bids. Other agents' mistakes, and changing behavior in general, make the assessment of expected

utility difficult.

If the agent receives no utility and cannot perceive the utility gradient, without further knowledge the algorithm updates bids in directions that decrease the amount of surplus conditioned on acceptance. Exploration in other directions does not tend to bear fruit. The reason search is limited to one direction is that with such a large search space, it is difficult to observe the benefits of updating in any other direction.

We would like an agent to have more information concerning the expected utility of a bid. To this end, we present two tools. We use the same heuristic that the host uses to pick bids to collapse an agent's search space. Additionally, we use a less local method to predict the acceptance of a bid and model auction participation as a series of Bernoulli trials. In the next section, we combine the two ideas, collapsing the search space and more accurate modeling of the possibilities, to derive an effective on-line bidding strategy.

6.6 An On-line Solution

In this section we restrict our attention to bidding strategies where the agent considers a bid to be repeated until the job at hand is completed. The restriction facilitates the computation of a bid's expected contribution to an agent's utility. Again we work with expectation-based utility. Bid selection requires the agent to assess the probability that a bid will be accepted by the host. The utility optimization will be more efficient and belief generation will be easier if we can compress the agent's search space.

6.6.1 Expected Utility

Let us focus our attention on a particular set of simple bidding strategies — ones where the agent picks a bid and repeats the bid until it completes its job at the current host. The expected utility of repeatedly submitting a bid to a host until the

job finishes is the sum, over the number of time blocks required to finish the job, of the products of the probability of the outcome taking the number of auctions and the utility of executing over that number of auctions. If the job requires a time blocks of allocation x, then the probability of finishing in fewer blocks is zero.

The value of a, given x, is

$$a := \left\lceil \frac{q_{i1}}{c_{i1} x_{i1} \Delta} \right\rceil \tag{6.18}$$

where Δ is the length of a time block.

The expected utility is then

$$\mathrm{E}[U(x,t)] = \sum_{j=a}^{\infty} p(x,t) \binom{j-1}{a-1} p(x,t)^{a-1} (1 - p(x,t))^{j-a} \left(\frac{E_i}{1 + h(j\Delta)} - a\Delta t \right) \tag{6.19}$$

where $p(x,t)$ is the probability that a bid wins the auction. We represent the possibility of winning an auction as an independent Bernoulli trial with probability $p(x,t)$. Equation 6.19 calculates the number of sequences that an agent could win $a-1$ of j auctions and the chance of each sequence. To avoid counting a sequence multiple times, the final auction in each sequence must be won, hence our use of sequences of $j-1$ auctions. We account for the final auction by multiplying the summand by $p(x,t)$.

In the context of this simplification of expected value, we use the shorthand

$$p := p(x,t) \tag{6.20}$$

$$q := 1 - p \tag{6.21}$$

From Equation 6.19, we factor out the common terms of $p(x,t)$ and $1 - p(x,t)$ to yield

$$\mathrm{E}[U(x,t)] = \left(\frac{p}{q}\right)^a \sum_{j=a}^{\infty} \binom{j-1}{a-1} q^j \left(\frac{E_i}{1+h(j\Delta)} - a\Delta t\right). \tag{6.22}$$

We then split the sum to represent the expected utility as a sum of two sums

$$\left(\frac{p}{q}\right)^a \sum_{j=a}^{\infty} \binom{j-1}{a-1} q^j \frac{E_i}{1+h(j\Delta)} \tag{6.23}$$

and

$$-a\Delta t \left(\frac{p}{q}\right)^a \sum_{j=a}^{\infty} \binom{j-1}{a-1} q^j. \tag{6.24}$$

Equation 6.24 is the agent's expected expenditure. We now focus on this part of the agent's expected utility and notice that the sum in Equation 6.24 is a hypergeometric function [GKP94]. Equation 6.24 expands to

$$\begin{aligned}
& -a\Delta t \left(\tfrac{p}{q}\right)^a \sum_{j=a}^{\infty} \tfrac{(j-1)!}{(a-1)!(j-a)!} q^j \\
= \quad & -a\Delta t \left(\tfrac{p}{q}\right)^a \left(\tfrac{q^a}{0!} + \tfrac{aq^{a+1}}{1!} + \tfrac{a(a+1)q^{a+2}}{2!} + \ldots + \tfrac{a\ldots(a-1+n)q^{a+n}}{n!} + \ldots\right) \\
= \quad & -a\Delta t p^a \left(1 + \tfrac{aq}{1!} + \tfrac{a(a+1)q^2}{2!} + \tfrac{a(a+1)(a+2)q^3}{3!} + \ldots + \tfrac{a\ldots(a-1+n)q^n}{n!} + \ldots\right) \\
= \quad & -a\Delta t p^a \left(\tfrac{1}{1-q}\right)^a \qquad [\mathrm{SR94}, \mathrm{F}(-a;1;1;q), \text{ p } 1066] \\
= \quad & -a\Delta t.
\end{aligned}$$

It is reassuring that the expected expenditure from the expected utility is what our intuition would lead us to — the cost of paying a bids of (x,t) for Δ time units each bid.

We attempt to solve the more difficult summand of the agent's expected utility. When the user has sharp expectations, high values of κ_i, we can approximate Equation 6.23 as

$$E_i \left(\frac{p}{q}\right)^a \sum_{j=a}^{b} \binom{j-1}{a-1} q^j \tag{6.25}$$

where

$$b := \lfloor \hat{\tau}_i / \Delta \rfloor \tag{6.26}$$

because the value of the sigmoid, $1/1 + h(j\Delta)$, is close to one for values of j less than $\hat{\tau}_i/\Delta$ and close to zero greater than that. We manipulate Equation 6.25 as follows

$$\begin{aligned}
& E_i \left(\tfrac{p}{q}\right)^a \left(\sum_{j=a}^{\infty} \tbinom{j-1}{a-1} q^j - \sum_{j=b+1}^{\infty} \tbinom{j-1}{a-1} q^j \right) \\
= \; & E_i \left(\tfrac{p}{q}\right)^a \left(\left(\tfrac{1}{1-q}\right)^a - \sum_{j=b+1}^{\infty} \tbinom{j-1}{a-1} q^j \right) && \text{Equation 6.24} \\
= \; & E_i \left(\tfrac{p}{q}\right)^a \left(\tfrac{1}{p^a} - \left(\tfrac{b!}{(a-1)!(b-a)!} q^{b+1} + \tfrac{(b+1)!}{(a-1)!(b+1-a)!} q^{b+2} + \dots \right.\right. \\
& \qquad \left.\left. + \tfrac{(b+n)!}{(a-1)!(b+n-a)!} q^{b+n+1} + \dots \right) \right) \\
= \; & E_i \left(\tfrac{p}{q}\right)^a \left(\tfrac{1}{p^a} - \left(\tfrac{a(a+1)(a+2)\dots bq^{b+1}}{(b-a)!} \left(1 + \tfrac{(b+1)1!q}{1!(b+1-a)} + \tfrac{(b+1)(b+2)2!q^2}{2!(b+1-a)(b+2-a)} + \dots \right.\right.\right. \\
& \qquad \left.\left.\left. + \tfrac{(b+1)(b+2)\dots nn!q^n}{n!(b+1-a)(b+2-a)\dots(b+n-a)} + \dots \right) \right) \right) \\
= \; & E_i \left(\tfrac{p}{q}\right)^a \left(\tfrac{1}{p^a} - \left(\tfrac{a(a+1)(a+2)\dots(b-1)bq^{b+1}}{(b-a)!\mathrm{B}(b+1,-a)} \int_0^1 \tfrac{z^b}{(1-z)^{a+1}(1-qz)} dz \right) \right) && [\text{SR94, p 1066}]
\end{aligned} \tag{6.27}$$

The function $\mathrm{B}(b+1, -a)$ is the standard Beta function [SR94, p. 957]. The evaluation of the expansion is messy, to say the least.

We switch tactics in from the evaluation of the expected utility to its approximation. The new approach approximates the utility lost from the agent losing a number of auctions. This new function has a domain that is the difference $b - a$, the number of auctions that an agent may lose and still yield positive utility, that is, the slack in the itinerary. We derive the function, $f(n)$, from Equation 6.25.

$$
\begin{aligned}
f(n=b-a) := \quad & E_i \left(\tfrac{p}{q}\right)^a \quad \sum_{j=a}^{b} \tbinom{j-1}{a-1} q^j \\
= \quad & E_i \left(\tfrac{p}{q}\right)^a \quad \sum_{j=a}^{b} \tfrac{(j-1)!q^j}{(j-a)!(a-1)!} \\
= \quad & E_i p^a \left(\tfrac{q^0}{0!} + \tfrac{aq^1}{1!} + \tfrac{a(a+1)q^2}{2!} + \tfrac{a(a+1)(a+2)q^3}{3!} + \ldots\right) \\
= \quad & E_i p^a \quad \sum_{k=0}^{n} \tfrac{q^k a(a+1)(a+2)\ldots(a+n-1)}{k!}
\end{aligned}
\tag{6.28}
$$

Fortunately, Equation 6.28 can be well approximated using a Taylor series expansion [Boa83]. We center the approximation about a number of lost auctions c for which it is easy to calculate the loss

$$
f(n) = \frac{f(c)}{0!} + \frac{(n-c)f'(c)}{1!} + \frac{(n-c)^2 f''(c)}{2!} + \frac{(n-c)^3 f'''(c)}{3!} + \ldots. \tag{6.29}
$$

One issue in computing the derivatives of Equation 6.28 is that $f(n)$ is valid only for integer values of n. To overcome this limitation, we modify the derivatives used in the Taylor series from the standard continuous derivative to the discrete derivative operator

$$
D[f(n)] := \frac{f(n+1) - f(n-1)}{2}. \tag{6.30}
$$

The second derivative can be expressed as

$$
\begin{aligned}
D[D[f(n)]] \quad & = \tfrac{D[f(n+1)] - D[f(n-1)]}{2} \\
& = \tfrac{f(n+2)-f(n)}{4} - \tfrac{f(n)-f(n-2)}{4} \\
& = \tfrac{f(n+2)-2f(n)-f(n-2)}{4}.
\end{aligned}
\tag{6.31}
$$

The relevant terms in approximating Equation 6.28 are

$$
\begin{aligned}
f(0) = &\quad E_i p^a \\
f(1) = &\quad E_i p^a (1 + aq) \\
f(2) = &\quad E_i p^a \left(1 + aq + \tfrac{a(a+1)q^2}{2}\right) \\
f(3) = &\quad E_i p^a \left(1 + aq + \tfrac{a(a+1)q^2}{2} + \tfrac{a(a+1)(a+2)q^3}{6}\right) \\
f(4) = &\quad E_i p^a \left(1 + aq + \tfrac{a(a+1)q^2}{2} + \tfrac{a(a+1)(a+2)q^3}{6} + \tfrac{a(a+1)(a+2)(a+3)q^4}{24}\right) \\
f'(1) = &\quad \tfrac{E_i p^a aq}{2} \left(1 + \tfrac{(a+1)q}{2}\right) \\
f'(2) = &\quad \tfrac{E_i p^a a(a+1)q^2}{4} \left(1 + \tfrac{(a+2)q}{3}\right) \\
f''(2) = &\quad \tfrac{E_i p^a aq}{4} \left(-1 - \tfrac{(a+1)q}{2} + \tfrac{(a+1)(a+2)q^2}{6} + \tfrac{(a+1)(a+2)(a+3)q^3}{24}\right).
\end{aligned}
\tag{6.32}
$$

To compute the first-order Taylor series approximation, we require the first derivative, so we center the approximation about the value 1.

$$
f(n) \approx E_i p^a \left(1 + aq + (n-1)\left(\frac{aq}{2} + \frac{a(a+1)q^2}{4}\right)\right) \tag{6.33}
$$

The second-order approximation requires more samples of $f(n)$, so we center the approximation around the value 2.

$$
\begin{aligned}
f(n) \approx &\quad E_i p^a \left(1 + aq + \tfrac{a(a+1)q^2}{2} + \tfrac{(n-2)a(a+1)q^2}{4}\left(1 + \tfrac{(a+2)q}{3}\right)\right. \\
&\quad \left. + \tfrac{(n-2)^2 aq}{4}\left(-1 - \tfrac{(a+1)q}{2} + \tfrac{(a+1)(a+2)q^2}{6} + \tfrac{(a+1)(a+2)(a+3)q^3}{24}\right)\right)
\end{aligned}
\tag{6.34}
$$

Figure 6.6 sketches an example of the computed expected utility to be gained from submitting a bid as a function of the gap in the number of auctions that the agent can lose and still meet its expected deadline. We plot the actual utility lost, as well as first and second-order Taylor series approximations. The approximations become wildly inaccurate after a certain point. Fortunately, we can determine reasonable bounds for accuracy. After such points we can approximate the loss function with the series' asymptotic value, E_i. For the first-order approximation, the switch is made once the estimation exceeds the asymptotic value. For the second-order approximation, we switch once the derivative of the approximation with respect to the auction gap is negative.

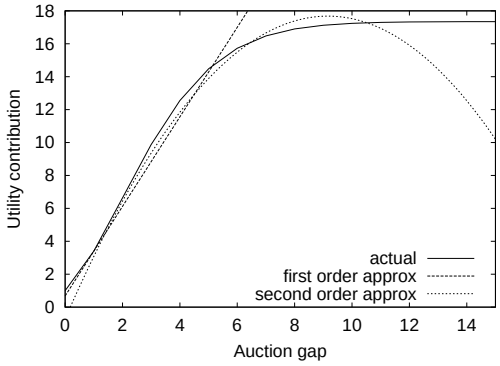

Figure 6.6: The contribution of consumption on expected utility as a function of the gap between the number of auctions required to complete a job and the maximum tolerance.

We reunite the two halves of the expected utility function and substitute the definitions of the minimum, a, and the maximum, b, number of auctions to arrive at

$$\mathrm{E}[U(x,t)] = f(\lfloor \hat{\tau}_i/\Delta \rfloor - \lceil q_{i1}/c_{i1}\Delta x \rceil) - \lceil q_{i1}/c_{i1}\Delta x \rceil \Delta t. \tag{6.35}$$

6.6.2 Density Heuristic

The optimization of Equation 6.35 requires that the agent has beliefs to formulate the probability of a bid's acceptance, $p(x,t)$. There is still a large two-dimensional space to search, however. We can reduce the bid search space by noticing that denser bids, that is, ones with high price-to-quantity ratios, are more likely to be taken. We formulate density as $\rho = t_i/x_i$, and determine which bid at any level of utility has the highest density. The idea is loosely derived from Gittens indices [Git89] and packing heuristics [Pis95]. We use the heuristic and search over a single dimension, either the range of densities, or over the bid size or price and let the density heuristic determine the other.

We commence this quest to prune the search space by examining the set of bids that yield the same utility. The function is just the expenditure as a function of bid size and utility

$$t_1(x,u) = \frac{E_i}{1+h(x)} - u - \sum_{j=2}^{N} t_{ij}. \tag{6.36}$$

Equation 6.36 defines an *isoquant,* a set of outcomes yielding identical utility, for the agent. We wish to find the set of all points that are tangent to the isoquant. The set is defined by

$$t_1(x,u) = \frac{\partial t}{\partial x} x. \tag{6.37}$$

Trivially, we substitute

$$\frac{\partial t}{\partial x} = -\frac{E_i}{(1+h(x))^2}\frac{\partial h}{\partial x} = \frac{E_i \kappa_i q_1 h(x)}{c_{i1} x^2 (1+h(x))^2} \tag{6.38}$$

into Equation 6.37 to get the set of bids that maximize bid density at any level of utility,

$$\rho(x) := t_1(x,u) = \frac{E_i \kappa_i q_1 h(x)}{c_{i1} x (1+h(x))^2}. \tag{6.39}$$

Figure 6.7 shows the points at that maximize density for several levels of utility. Figure 6.8 plots the resulting set of points that maximizes bid density for every level of utility. We will shortly use this maximization to plan a bidding strategy and expenditure-planning algorithm.

Figure 6.9 plots an example of how the density heuristic changes over time. The maximum-density bidding space becomes smoother as the agent waits, or is denied service. The change in the height of the curve reflects the fact that the agent has less to gain as its deadline becomes more difficult to achieve. The urgency of the job is represented by the fact the agent is willing to pay for large quantity bids.

For the density heuristic $\rho(x)$ to be useful for bidding, it should be the case that as we increase x, the agent's heuristically optimal bid monotonically becomes less dense. This condition is identical to

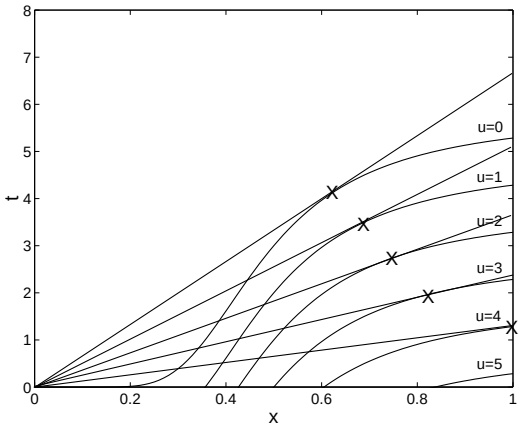

Figure 6.7: The points along several isoquants that maximize bid density with fixed utility.

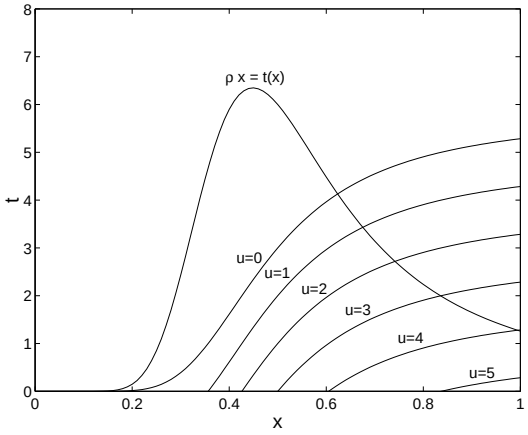

Figure 6.8: Illustration of points satisfying density heuristic.

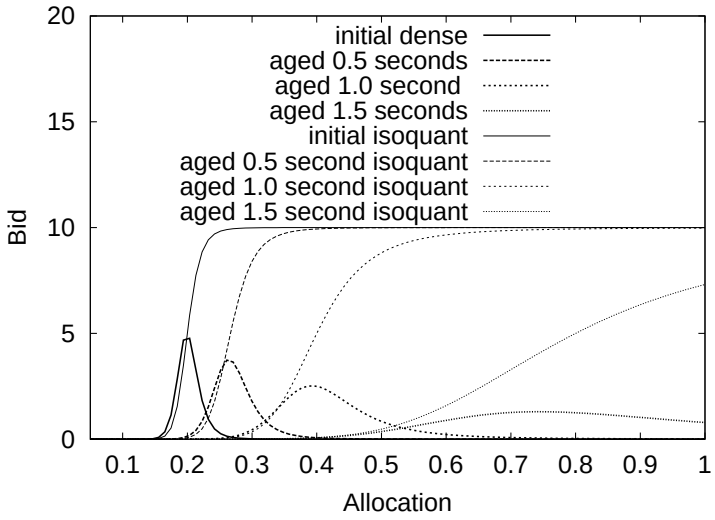

Figure 6.9: A demonstration of how an agent's optimal bid density function and zero utility isoquant change over time.

$$\frac{\partial \rho/x}{\partial x} \leq 0, \tag{6.40}$$

which we now prove.

Theorem 4 *The bidding heuristic $\rho(x)$ is monotonically density decreasing as x increases for bids that complete the itinerary in time bounded by the expectation.*

Proof of Theorem 4

The rate of bid density change with respect to x is

$$\frac{\partial \rho/x}{\partial x} = \frac{-2h(x)(1+h(x))x - kh(x)(1-h(x))}{x^4(1+h(x))^3}, \tag{6.41}$$

where

$$k := \kappa_i q_{i1}/c_{i1}. \tag{6.42}$$

For the agent to meet its performance expectation,

$$\sum_{j=1}^{M_i} \frac{q_{ij}}{c_{ij}x_{ij}} \leq \sum_{j=1}^{M_i} \hat{\tau_{ij}} \tag{6.43}$$

must hold. This condition implies that $h(x_{i1})$ is at most one. Since $h(x)$ and k are positive, Equation 6.41 and the condition imply that $\partial(\rho/x)/\partial x$ is negative. $\diamondsuit$

Theorem 4 allows us to use the density heuristic for expectation-based utility and infer the price of computation at a host.

6.6.3 Bidding Beliefs

Now that we have compressed the effective bidding space, we show how an agent can project a history of accepted and rejected bids onto the space and build a belief function that maps a bid to the perceived probability that the host will accept the bid.

Again, we are inspired by existing work. Gjerstad and Dickhaut [GD98] construct belief functions that they use for price formulation in double auctions. Buyers and sellers must derive an expected utility function of a bid or ask to be sent to the market. Gjerstad and Dickhaut only consider transactions that involve single objects. Hence, their belief function need only operate on a single domain. Their belief function is a monotonic map of price to probability.

In the previous section, we collapsed our problem's bidding space onto a single dimension. We create a map from a bid, to a point on our bidding space, characterized by its quantity request, to the probability that it is accepted. The idea is that given any bid (x, t), we can find a bid that is at least as dense and will yield the same utility conditioned on acceptance. We can find the bid by numerically solving the program

$$\begin{array}{rl} \text{solve} & t' = \rho(x') \\ \text{such that} & t'/x' \geq t/x, \\ & U_i(x,t) = U_i(x',t') \end{array} \tag{6.44}$$

to yield a bid that is heuristically more likely to be accepted by the host. We can

quickly solve the program in Equation 6.44 with standard numerical root-finding routines [PTVF92].

Given a collection of bids submitted to a host and their acceptance, we can build a belief function that estimates the probability that any bid will be accepted. We construct the belief function through maximization of the likelihood function

$$L(\beta_0, \beta_1 | X) = \log \left(\prod_{x_i \in \text{taken}} p(x_i, \beta_0, \beta_1) \prod_{x_j \notin \text{taken}} 1 - p(x_j, \beta_0, \beta_1) \right), \tag{6.45}$$

where here, $p(x)$ is the estimated likelihood that the host will take the bid,

$$p(x, \beta_0, \beta_1) := \frac{\exp(\beta_0 x + \beta_1)}{1 + \exp(\beta_0 x + \beta_1)}. \tag{6.46}$$

We represent the likelihood that a single bid is accepted as a sigmoid with parameters β_0 and β_1 that adjust the position and sharpness of the sigmoid. Intuitively, $-\beta_1/\beta_0$ is the cutoff bid for the agent. Bids along $\rho(x)$ with x larger than that ratio are unlikely to be accepted. The parameter β_0 determines how quickly the probability falls off. Again, we use numerical methods to maximize the likelihood function [HL89, PTVF92]. We use the sigmoid in form from Equation 6.46, rather than the form from the utility function in Equation 6.4 to avoid round-off error in maximization.

Figure 6.10 shows an example of a fitted belief function

Qualitatively, we observed in our simulations that agents' belief functions are sharp; the gap between the quantity that agents map accepted and rejected bids is narrow. This quality of belief functions is due to the almost angular nature of most agents' maximal density function, $\rho(x)$. Frequently, an agent's function $\rho(x)$ will map closely to zero for all but a narrow domain of x. Fudenberg and Levine [FL98] observe that smoothing the belief function encourages exploration. We will explore the immediate utility cost versus the benefit of accelerated learning from smoothing the belief function in future work.

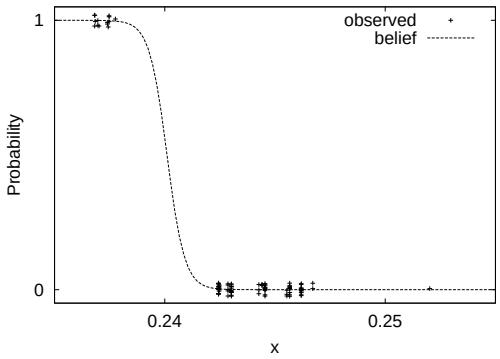

Figure 6.10: The constructed belief function and mapped input. We have slightly permuted the observed probability values of the bids to represent the frequency of observed bids mapped to a common bid quantity.

6.6.4 The Algorithm

Sections 6.6.1–6.6.3 provide the three ingredients that we now use to create a bidding strategy. An agent examines the history of bids at a host; maps the bids to its own density heuristic; and maximizes the expected utility using a conjugate-gradient method [PTVF92]. Algorithm 6 presents the steps that an agent uses to calculate its bid.

To compute its expected utility, an agent must estimate the costs and latencies incurred in executing its future jobs. So as to lessen the impact of the agent's actions on its estimations, if the agent has plans repeat its visit to its current host, we append the job to be completed in the repeat visit to the current job. Thus the agent optimizes a longer series of auctions.

For other jobs, we assume that each host announces the bids that it accepts. The agent uses its user's latency expectation and computes the cost of the job using the average cost-per-computational-portion to meet the user's expectation.

Algorithm 6 Density-heuristic bid

variable uses	
accepted	A list of accepted bids.
rejected	A list of rejected bids.
$x_{\min}$	The smallest bid that can yield positive utility.
x_p	The amount to shift the fitted belief sigmoid.
$x_\#$	The precision of the fitted belief sigmoid.

$x_{\min} \leftarrow$ smallest bid on $\rho(x)$ with non-negative utility
for each point i in history **do**
  $j \leftarrow$ map-point(i) {map history to density-maximized equal-utility}
  if $j.x < x_{\min}$ **then**
    continue
  else if i accepted **then**
    accepted.push (j)
  else
    rejected.push (j)
  end if
end for
if accepted.size () = 0 and rejected.size () = 0 **then**
  submit bid with uniformly random density from $(0, \rho(x_{\min})/x_{\min})$ {no data– bid arbitrarily}
else if rejected.size () = 0 **then**
  $i_{\max} \leftarrow$ largest quantity bid in accepted {bump up the largest bid}
  submit bid $((1 + i_{\max}.x)/2, \rho((1 + i_{\max}.x)/2))$
else if accepted.size () = 0 **then**
  $i_{\min} \leftarrow$ least quantity bid in rejected {search for a smaller bid}
  submit bid $((x_{\min} + i_{\min}.x)/2, \rho((x_{\min} + i_{\min}.x)/2))$
else
  $(x_p, x_\#) \leftarrow$ max-log-likelihood fit {search!}
  search on $((x_{\min}, \rho(x_{\min})), (1, \rho(1)))$ for utility maximizing bid
  submit results of search
end if

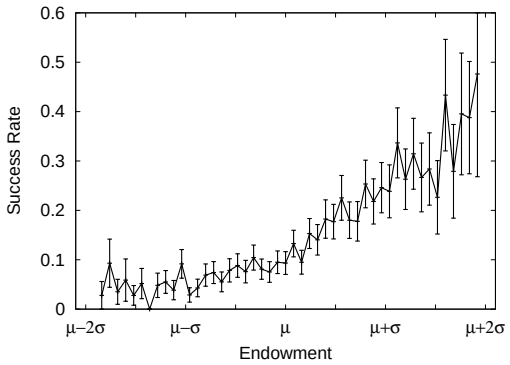

Figure 6.11: Agent endowment relative to job size versus success rate in an over-constrained system.

6.7 Simulation

We implemented a network of hosts, connected by a network with topology that we generated with GT-ITM [CZ96], inside the ns-2 [ns01] simulator. Because of the overhead involved in agents' learning and the use of Tcl to drive the ns simulator, we simulated a network only of size 12. As in Chapter 5, we drew hosts' capacities from a truncated Gaussian distribution. Agents had exponentially distributed numbers of jobs to complete, and each job's size was Pareto distributed.

We ran an experiment to verify that endowing an agent with more currency increased its chances to complete its itinerary similar to the experiments plotted in Figures 5.5 and 5.6. Figures 6.11 and 6.12 plot the effect of endowment on agents' chances of success. Like the experiments that Figures 5.5 and 5.6 represent, the distinguishing feature between the experiments plotted by Figures 6.11 and 6.12 is the scale of the computational needs of the agent population.

Figure 6.11 plots the relationship between endowment and success when the aggregate demand exceeded system capacity by 50%; in Figure 6.12, demand was 75% of capacity. The differences between the results of the two experiments were minor. Agents performed better with less congestion, but hosts were not able to discriminate between high and low priority agents the same way that they did in Chapter 5; some

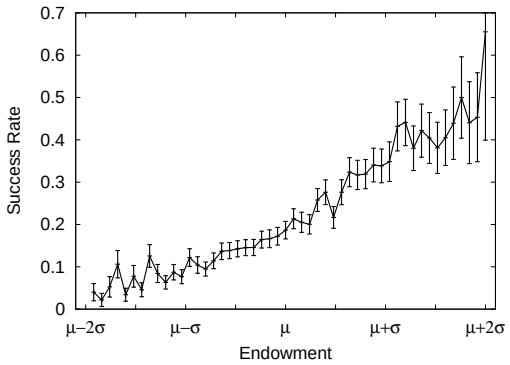

Figure 6.12: Agent endowment relative to job size versus success rate in an system running at 75% capacity.

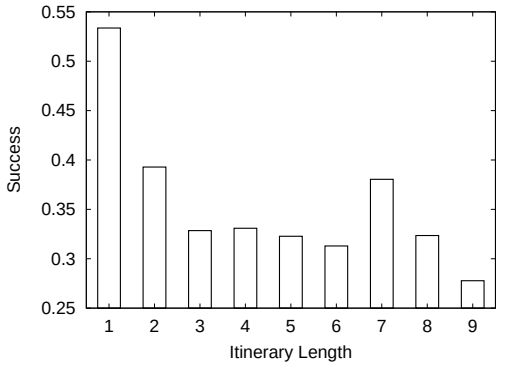

Figure 6.13: The observed probability of an agent completing its itinerary versus the length of its itinerary.

low priority agents still computed, rather than opting to wait.

Part of the reason that endowment had less of an effect on performance in the revenue maximization model than in the shared-resource-ownership model in Chapter 5 is that the number of jobs in an agent's itinerary also affected its ability to complete. Figure 6.13 plots the relationship between the observed probability that an agent completed its itinerary and the number of jobs in the agent's itinerary. The mean number of jobs in agents itineraries was three.

In the context of host-revenue maximization, agents' requests do not directly translate to utilization as in Chapter 5. In this chapter's model, a host has some value in not selling computation. Agents spend time learning hosts' values for com-

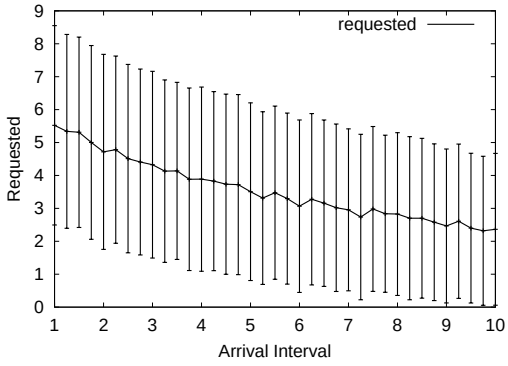

Figure 6.14: Agent arrival interval versus mean computational requests. The theoretical system capacity could support an arrival every six time units.

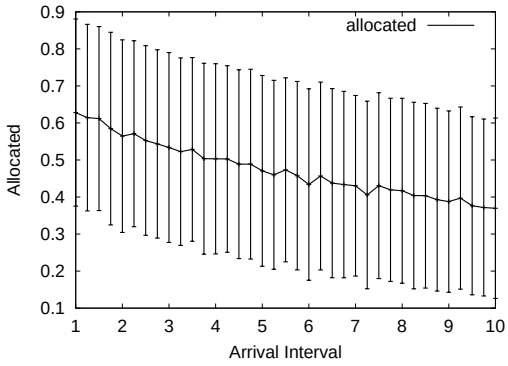

Figure 6.15: Agent arrival interval versus mean computational allocation. The theoretical system capacity could support an arrival every six time units.

putation as well as the extent of the competition for computation. Figure 6.14 shows how agents' requests dominated system capacity when the system could theoretically handle six arrivals per time unit. Figure 6.15 shows the amount of computation that the system allocated.

Agents' resource requests scaled linearly as more agents entered the system, as we illustrate in Figure 6.14, but we observed that agents learned that the quality of service that the hosts provide suffered as more agents entered the system. The learning was evident in hosts' constant ability to capture about the same portion of agents' endowments across all arrival intensities, which we plot in Figure 6.16. Agents were fairly compensated in that they received roughly the same utility, relative to the

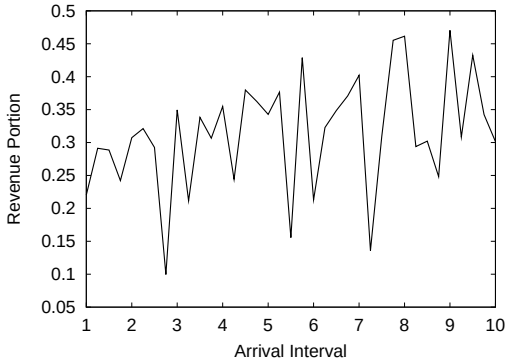

Figure 6.16: Agent arrival interval versus mean agent expenditure. The theoretical system capacity could support an arrival every six time units.

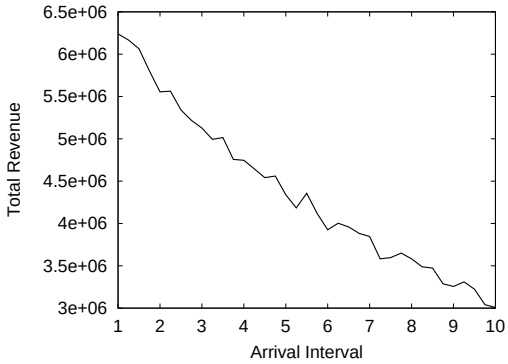

Figure 6.17: Agent arrival interval versus payments made by agents. The theoretical system capacity can support an arrival every six time units.

optimal utility, across all arrival intensities.

The constancy of the revenue portion capture is a positive factor for hosts. It shows that the marginal revenue for each additional agent in the network is positive, even as the quality of service that a host provides diminishes. We plot how hosts' revenues increased with the fall in inter-arrival times in Figure 6.17.

6.8 Conclusions

In this Chapter we take a broader view to market-based computational scheduling than in Chapter 5. We drop the assumption that hosts and agents share objectives

and ownership of resources as well as the assumption that an agent may not return its unspent endowment to its user. The value for savings affects our choice of utility function, which impacts our ability to express resource-access priority through an agent's endowment. We believe that the model applies to more resource-allocation problems than the one in Chapter 5, but we do note that there are costs involved in separating hosts and agents' interests.

To incorporate the value of savings to an agent, we adopt a quasi-linear utility function. We justify the use of simple utility since measuring an agent's value for future consumption is difficult. The principle drawback of quasi-linear utility functions is that they exhibit no wealth effect and, hence, do not influence an agent with a larger endowment to compute more quickly. A user with an intent of having richer agents complete faster than poorer ones must imbue each agent with a performance expectation that matches its endowment. The drawback of this approach is that it requires extra computation when the agent is created and the agent is inflexible to the desires of its user when the state of the network changes. The agent is interested in receiving a specific quality of computation, which justifies the use of our simple, first-price, single-bid, repeated auctions.

An agent journeys through the network attempting to maximize its savings and performance without assurance from its hosts that high performance at one point in time indicates comparable performance in the future. A host and agent in our simulation make independent transactions, though the agent's utility depends on the entire series of transactions. This fact, and the separation of host and agent objectives in general, give rise to incentive for the agent to withhold information regarding its intent. Our experiments show the effect in agents' reluctance prioritize themselves as neatly as they do in Chapter 5.

Without a wealth effect, an agent is interested in a specific amount of computation to fulfill its expectation, and has less incentive to divulge preferences. On the other

side of the market, the host has no rationale to publish its costs and agents can waste precious time learning what bids the host will accept. An agent can mitigate the time spent learning through extrapolating from bidding history transactions the probability of a bid's acceptance. This illuminates the model's sensitivity to estimation errors on the part of both agents and hosts.

The motivations guiding system design in this chapter, however, are not system efficiency, but revenue maximization for the host. We motivate resource owners to provide service to general agents. This incentive enables computing in distributed environments where users and resource owners do not necessarily trust each other or, more generally, where there is a cost of providing service to users.

Our system is effective in capturing agents' endowments. While agents do learn to scale back their bids as congestion increases and quality of service declines, we observe empirically that the marginal revenue to a host of the arrival of another agent is positive.

Furthermore, the allocation policy that we present makes few assumptions on the nature of agents' utilities. For reasonable numbers of agents, the allocation process runs quickly, and in the worst case, the host can sort bids by price per-unit of computation and choose bids in a greedy fashion.

a	The number of auctions that the agent must win to complete its current job at the host.
b	An approximate upper bound on the number of auctions in which an agent can participate and receive positive utility.
$D[f(n)]$	The derivative of the discrete function $f(n)$.
Δ	The length of time for which the host guarantees computation to auction winners.
ϵ	The approximation gap that the host uses for bid selection.
$h(x)$	Shorthand for $\exp(\kappa_i \sum_{j=1}^{M_i} q_{ij}/c_{ij}x_{ij} - \hat{\tau}_i)$.
κ_i	The precision of a user's expectation that the i-th agent completes in time less than $\hat{\tau}_i$.
$p(x,t)$	The believed probability that a bid for amount x at price t will be accepted by the host.
$\rho(x)$	The density-maximized bid of size x.
$q(x,t)$	The complement of $p(x,t)$.
s	The scaling factor used by the host to pack the big bids.
$\hat{\tau}_i$	The latency that the owner of the i-th agent expects the agent to incur.
U_0	The host's utility.
v_0	The host's marginal valuation of the resource.
$v*$	The host's reserve price for computation.

Table 6.1: Notation introduced in Section 6.

Chapter 7

Itinerary Planning

In Chapters 5 and 6, we explore how an agent can budget its expenditures and participate in auctions to acquire resources given that it already has a route planned. We avoid the problem of an agent's choice of route through the assumption that the agent plans to visit an "average" host to complete each task in its itinerary. In part, we take the approach to avoid a circular problem: an agent needs a route to plan its expenditures, but performance and budget considerations effect the choice of route. For some applications, the route selection problem may be irrelevant or too cumbersome to consider; the agent may not have a solid itinerary, or the ability to substitute hosts may be unclear.

In this chapter, we discuss how an agent with a budget constraint can choose its route. The route formulation deals with a special type of n-partite graphs. Optimal route construction with known costs and latencies is a special instance of the shortest constrained-path problem, which is NP-complete even under the restricted graph topology. In Section 7.1 we show the graph representation of an agent's itinerary. We show the reduction from the knapsack problem in Section 7.2 to justify the hardness of route formulation. We present algorithms to construct routes in Section 7.4 and examine their performance in Section 7.5.

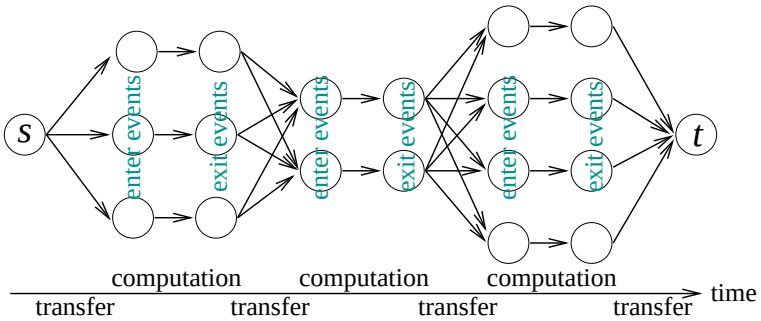

Figure 7.1: An example route-possibility graph for an agent with three tasks.

7.1 Route Representation

An agent's possible routes can be represented with a restricted type of directed n-partite graph. An n-partite graph partitions its vertices into n sets. The vertices of the graph represent the agent choosing to execute at a host (an "enter" event) or leave a host (an "exit" event). The edges from enter to exit events represent time spent computing at a host. The edges that connect exit to enter events represent network latency. The edges can have costs associated with them, as well as latencies. Figure 7.1 presents an example route-possibility graph. In this chapter we only consider costs from enter to exit nodes, but the work easily extends to the more general case where all nodes have costs.

7.2 Shortest Constrained-Path Problem

The selection of the shortest path through a graph with both edge lengths and edge costs where the summed costs of the edges in the path may not exceed a pre-selected constant is the shortest constrained-path problem (SCP). Even in our restricted domain of n-partite graphs, the problem is NP-complete and we now reproduce a reduction from [AMO93] to show the problem's difficulty.

Theorem 5 *An agent's routing problem is NP-complete [AMO93].*

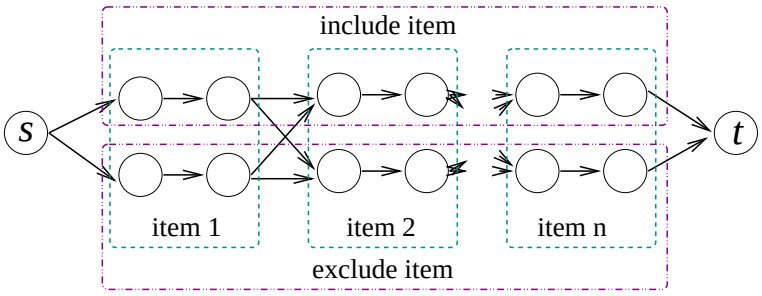

Figure 7.2: A recipe for expressing any 0-1 knapsack problem as an instance of our routing problem.

Proof of Theorem 5

We show that we can map any knapsack problem [GJ79] to our host selection problem. The knapsack problem involves picking a subset of items, each with a weight and value, that maximizes the value carried by a container with a fixed weight constraint capacity. The knapsack problem is NP-complete.

We can express any knapsack problem as an agent routing problem through the following construction. Represent every item in the knapsack as a task in the agent's itinerary. To complete a task, the agent may visit a host that charges the weight of the item and will complete the task in zero time. This represents the inclusion of the object in the knapsack. Alternatively, the agent may complete the task with zero cost in time equal to the value of the object. This represents excluding the object from the knapsack. The edges connecting exit to enter events have zero latency. Figure 7.2 illustrates the structure of the reduction.

The cost incurred by an agent in the course of its itinerary represents the weight of the objects placed in the knapsack. The end-to-end latency of the itinerary represents the value not carried by the knapsack. Minimization of the excluded value is identical to maximization of the value of the objects in the knapsack.

Since every instance of the knapsack problem can be represented by our agent-routing problem and a solution to the routing problem can be verified in polynomial time, the agent-routing problem is NP-complete. $\diamondsuit$

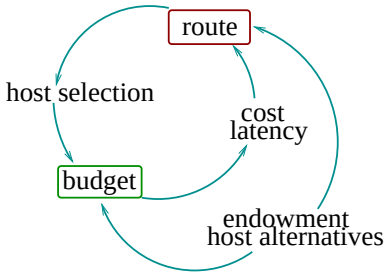

Figure 7.3: The recursive dependency of the host selection and budgeting problems.

7.3 Selection Versus Budgets

An agent's route selection problem is generally more involved than a shortest constrained-path problem. To complete its itinerary in either Chapter 5 or 6, an agent must negotiate the cost and the latency incurred at each host after selecting the host. The budgeting process requires knowledge of the hosts to be visited, but host selection requires an estimate of the cost and performance of visiting a host. Thus route formulation is more difficult than the shortest constrained-path problem.

A method that guarantees that a chosen route is feasible estimates the cost and performance of visiting one host based upon an itinerary of visiting hosts with the highest congestion and lowest capacity for each service the agent consumes. With the pessimistic cost and performance estimate the agent can select a route from the network using any of the strategies that we will present in Sections 7.4.

A more moderate solution is to use the expected capacities and congestions for estimates of the hosts to be visited. This solution does not guarantee a feasible solution, but works in practice. Using this heuristic, an agent can choose an itinerary of fast, expensive hosts with the expectation at choosing each that the individual host selected is an anomaly in the entire itinerary. The estimation method works in practice because the priority negotiation policies with which are concerned allow the agent to tune its local consumption to match global performance expectations. That is, there is slop between the routing and budgeting stage in itinerary formulation.

In our implementation of the algorithms below, an agent first applies the mean cost and performance estimations to the host-selection problem. If the agent cannot find a feasible path, the agent applies the worst-case estimates. After the agent chooses a route, the it applies the budgeting algorithm derived from Equation 5.10 to the route to allocate its expenditures.

7.4 Solutions

We next present three solutions to our routing problem: the greedy solution that we implemented in Chapters 5 and 6, a heuristic algorithm inspired from linear programming, and an algorithm that computes the optimal shortest constrained-path given an agent's estimates. To compare each algorithm, we simulate the host-selection algorithms inside a network that uses the allocation policy that we develop in Chapter 5.

7.4.1 Greedy Solution

We revisit the routing solution that we use for the simulations in Chapters 5 and 6. For every task, the algorithm constructs a budget for every host alternative. The algorithm reserves a budget portion for the tasks already selected and formulates a new sub-budget for the remaining tasks. In this manner, the process refines the budget with each task that it considers. Algorithm 7 defines the operation. It requires computation linear in the number of hosts to be considered.

7.4.2 The Dual Heuristic

Our second method for host selection is inspired by linear programming. The algorithm generates a linear program that represents a simplified host selection problem. The advantage of the simplified program is that we can solve the dual of the linear program without solving the original linear program. The optimization of the dual program provides a good solution to host selection with minimal computation.

Algorithm 7 Greedy host selection for agent i

variable uses	
$host$	A pointer to the best host.
E'	Budget remaining estimate.
j	Task iterator.
k	Host choice iterator.
l	The agent's location.
t_j	The expected expenditure for the j-th job.
$\tau_{\min}$	The lowest expected latency for a job.
τ_{est}	The expected latency of executing at a host.
$visit$	The list of hosts to visit.

1: $l \leftarrow$ current location
2: $E' \leftarrow E_i$
3: **for each** task j **do**
4: $\quad \tau_{\min} \leftarrow \infty$
5: $\quad$ **for each** next host alternative k **do**
6: $\quad\quad \tau_{est} \leftarrow$ expected time at k with budget E' + travel time from l to k
7: $\quad\quad$ **if** $\tau_{est} < \tau_{\min}$ **then**
8: $\quad\quad\quad \tau_{\min} \leftarrow \tau_{est}$
9: $\quad\quad\quad host \leftarrow k$
10: $\quad\quad\quad t_j \leftarrow$ expense at k
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad l \leftarrow host$
14: $\quad E' \leftarrow E' - t_j$
15: $\quad visit$.push_back(l)
16: **end for**
17: **return** $visit$

Linear-Programming Representation

We can construct a simple linear program to approximate an agent's computation time by ignoring transfer time between hosts. The agent has an endowment of E to complete M tasks. Each task can be executed at one of K alternative sites. The estimate expenditure of completing the j-th task at the k-th host alternative is e_{jk} and the agent estimates that the task and transfer time to any other host from the jk-th will take τ_{jk} seconds. The linear program is then

$$\begin{aligned} & \min_{x_{jk}} \sum_{j=1}^{M} \sum_{k=1}^{K} x_{jk}\tau_{jk} \\ \text{Such that} & \\ & \sum_{k=1}^{K} x_{jk} \geq 1 && \forall j \in [1 \ldots M] \\ & x_{jk} \geq 0 && \forall j \in [1 \ldots M], k \in [1 \ldots K] \\ & \sum_{j=1}^{M} \sum_{k=1}^{K} x_{jk} e_{jk} \leq E. \end{aligned} \tag{7.1}$$

The program optimizes the host choice for each task. The indicator variables, x_{jk}, represent the agent's choice to execute its j-th task at the k-th host choice. An ideal solution assigns the value zero to all hosts that the agent does not visit and one to the indicators that represent the hosts that the agent does visit. The linear program relaxes the problem by assigning fractional values to the indicators. The linear program optimizes a continuous problem that can be used to approximate a solution to the discrete problem. This technique is used to derive approximation algorithms for many NP-complete problems [MR97]. Possible implementations could have the agent choose to visit the host with the highest weight or choose the jk host with probability x_{jk}. For the moment, we ignore the choice of hosts and concentrate on the overhead of solving the linear program. A common method of reducing the complexity of solving the linear programming is formulating the linear problem as the dual problem [GW97].

Every linear minimization program has a dual maximization program [Chv83].

While the original linear program, the primal problem, minimizes the objective function over the constraint space, the dual problem maximizes another objective function over a set of new constraints derived from the original problem's variables. Specifically, if the linear program is

$$\begin{aligned} & \max_{x_j} \sum_{j=1}^{n} c_j x_j \\ \text{Such that} & \\ & \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i \in [1 \ldots m] \\ & x_j \geq 0 \qquad\quad \forall j \in [1 \ldots n], \end{aligned} \tag{7.2}$$

then the dual formulation is

$$\begin{aligned} & \min_{y_i} \sum_{i=1}^{m} b_i y_i \\ \text{Such that} & \\ & \sum_{i=1}^{m} a_{ij} y_i \geq c_j \quad \forall j \in [1 \ldots n] \\ & y_i \geq 0 \qquad\quad \forall i \in [1 \ldots m]. \end{aligned} \tag{7.3}$$

Intuitively, the formulation of the dual exchanges the roles of the variables and the constraints. The Strong Duality Theorem states that the two problem's optimal solutions are identical if a bounded feasible solution for either one of the problems exists [Chv83]. Solving the dual problem is useful if the number of constraints is larger than the number of variables, since the dual problem yields a lower dimensional search space over which to optimize.

The dual formulation for our linear program formulation of our simplified routing problem in Equation 7.1 is

$$\begin{aligned} & \max_{y_k} -E y_{M+1} + \sum_{j=1}^{M} y_j \\ \text{Such that} & \\ & y_j - e_{jk} y_{M+1} \leq \tau_{jk} \qquad \forall j \in [1 \ldots M], k \in [1 \ldots K] \\ & y_j \geq 0 \qquad\qquad\quad \forall j \in [1 \ldots M+1]. \end{aligned} \tag{7.4}$$

Since each y_j for j at most M is constrained to be at most $e_{jk}y_{M+1} + \tau_{jk}$, we can simplify Equation 7.4 from a program of $M+1$ variables to one of a single variable,

$$\max_{y_{M+1}} \left(-Ey_{M+1} + \sum_{j=1}^{M} \min_{k \in [1 \ldots K]} (y_{M+1}e_{jk} + \tau_{jk}) \right). \tag{7.5}$$

The solution to Equation 7.5 provides an upper bound on the agent's simplified performance since the optimal fraction solution is at least as good as the integer solution and the fractional problem's domain includes the integral problem's domain.

We can solve the original linear program through optimization of the dual along a single dimension, y_{M+1}. With the optimal value of y_{M+1} we can choose pairs (j, k) such that the constraint represented by the variable y_j is tight for the pair.

When there are more than one pair for which the constraint is tight, the optimization represents choosing a mixture of the hosts represented by the pairs. It must be the case that the set of choices represented by the cheapest hosts for which the constraints are tight is a feasible itinerary. This leads us to the following heuristic that we define in Algorithm 8.

The algorithm's run time is bounded by the maximization of the single dimensional dual function. In theory, the primal and the dual linear-programming problems can be solved in time polynomial to the number of constraints. In our implementation, we use Newton's method for optimization of line 5 of the algorithm to yield good practical performance.

7.4.3 Lagrangian Relaxation

To compare the performance of our algorithms, we present an algorithm that optimally solves the shortest constrained-path problem [AMO93] using our cost estimations. Once the algorithm selects a route, we apply the budgeting process to allocate the agent's expenditures across its itinerary.

The algorithm uses Lagrangian relaxation to pose the shortest constrained-path

Algorithm 8 Dual-heuristic host selection

variable uses	
j	The task iterator.
k	The host alternative iterator.
T	The set of hosts with cheapest composite cost.

1: **for each** host alternative k **do**
2: estimate cost and execution time of visiting k holding other host choices at "average"
3: generate coefficients for k
4: **end for**
5: $\max_{y_{M+1}} Ey_{M+1} + \sum_{j=1}^{M} \min_{k \in [1 \ldots K]} (y_{M+1} e_{jk} + \tau_{jk})$
6: **for** $k \in [1 \ldots K]$ **do**
7: $T \leftarrow \arg\min_j \{y_{M+1} e_{jk} + \tau_{jk}\}$ choose cheapest host with tight constraint
8: select $j : \arg\min_j \{e_{jk} : j \in T\}$
9: **end for**

problem as a series of shortest-path problems. We modify the length of each arc in the itinerary graph to be a linear combination of the latency and cost incurred in choosing the arc as part of the agent's path. Thus the composite length of an arc connecting the j-th and k-th nodes in the itinerary is

$$l_{jk} := \mu t_{jk} + \tau_{jk}, \tag{7.6}$$

where t_{jk} is the monetary cost leveled on the agent for selection of the arc and τ_{jk} is the time delay incurred in choosing the arc.

Assignment of μ with the value of zero corresponds to a shortest-path problem where the agent chooses the fastest path through the network. The limit of μ increasing is equivalent to the agent's selection of the least expensive path.

We search for a value of the Lagrangian multiplier, μ, that maximizes the shortest composite path through the agent's itinerary graph. The process proceeds by iteratively selecting a value for μ and solving a shortest path problem. The composite cost of the shortest path is our potential function.

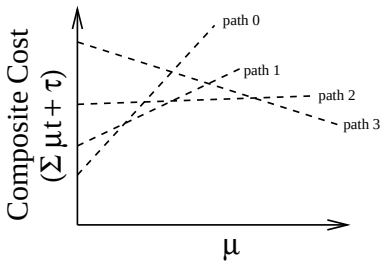

Figure 7.4: The search space for maximization of the Lagrangian potential function, expressed as the shortest composite path through the graph.

Figure 7.4 shows the search space for the Lagrangian relaxation problem. The space is bounded by lines that represent the composite cost of each path in the graph. We use linear interpolation of the two best known paths to maximize the composite cost. In the worst case, however, we would have to consider every path in the graph and the number of paths is exponential in the number of tasks to consider. Thus, the worst-case run time of the algorithm is exponential in the number tasks.

Once we maximize the potential function, we choose the shortest of the feasible paths represented by the lines that intersect at the maxima. Algorithm 9 shows the operation of the search.

7.5 Simulation

We simulated the performance of agents that used each host-selection algorithm in the context of the allocation framework from Chapter 5. We used a GT-ITM network topology with 100 hosts. Agents had exponentially distributed numbers of tasks and the size of each was Pareto distributed. For each task, an agent considered ten host alternatives.

Figure 7.5 plots the execution times of the three different algorithms when agents have itineraries with mean length ten. [1] As expected the order of the run times from

[1] We ran the simulation on a single 450 megahertz Pentium III processor running Linux kernel version 2.2.16-22.

Algorithm 9 Lagrangian relaxation method for solving the shortest constrained-path problem in agent host selection.

variable uses	
b	Intercept of linear estimate of composite-path cost.
k	Host choice iterator.
$\partial\phi$	The change in composite-path cost with respect to μ.
subscript H	Parameters at the upper bound of μ.
subscript L	Parameters at the lower bound of μ.

1: **for each** host alternative k **do**
2: estimate cost and execution time of visiting k holding other host choices at "average"
3: generate coefficients for k
4: **end for**
{Initialize upper bound for μ}
5: $\mu_H \leftarrow$ pathLengthBound()/($E -$ pathCost($\mu \to \infty$)) {for cheapest cost path.}
6: $\phi_H \leftarrow$ pathLength(μ_H) $- \mu_H E$
7: $b_H \leftarrow \phi_H - \mu_H \partial\phi_H$
8: $\partial\phi_H \leftarrow$ pathCost(μ_H) $- E$
{Lower bound is fastest-path}
9: $\mu_L \leftarrow 0$ {value for μ.}
10: $\phi_L \leftarrow$ pathLength(μ_L) $- \mu_L E$
11: $b_L \leftarrow \phi_L - \mu_L \partial\phi_L$
12: $\partial\phi_L \leftarrow$ pathCost(μ_L) $- E$
{Use linear-interpolation search over μ.}
13: **while** $\mu_H > \mu_L$ **do**
14: $\mu \leftarrow$ solution to $\mu\partial\phi_L + b_L = \mu\partial\phi_H + b_H$
15: $\phi \leftarrow$ pathLength(μ) $- \mu E$
16: **if** $\phi > \phi_L$ **then**
17: $\mu_L \leftarrow \mu$
18: $\phi_L \leftarrow \phi$
19: $b_L \leftarrow \phi - \mu\partial\phi$
20: $\partial\phi_L \leftarrow$ pathCost(μ) $- E$
21: **else**
22: $\mu_H \leftarrow \mu$
23: $\phi_H \leftarrow \phi$
24: $b_H \leftarrow \phi - \mu\partial\phi$
25: $\partial\phi_H \leftarrow$ pathCost(μ) $- E$
26: **end if**
27: **end while**
28: **return** edges along shortest path with edge lengths $l + \mu t$

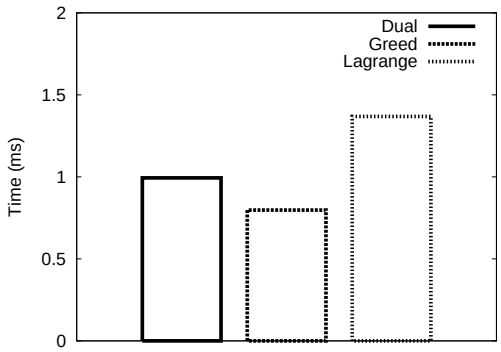

Figure 7.5: Mean execution times in seconds of the three host selection algorithms for itineraries with mean ten hops.

best to worst was greedy, dual, Lagrange, though it was the case that solving the Lagrangian shortest-constrained-paths problem was still reasonable when the agent selected from 100 hosts. We applied a t-test to determine that the execution times of the greedy and Lagrange were different from dual algorithm's execution time with a better than 5% level of significance.

Figure 7.6 shows the performance of the three algorithms. We measured performance by the ratio of the minimum latency possible and the latency the agent incurred in routing its itinerary. The ratio yields a metric between one and zero. One is an upper bound on the optimal performance and zero means that the algorithm could not find a feasible path. Note that the performance upper bound only connotes system capacity, not necessarily that there is a feasible path that the agent can take to achieve that latency.

We represent mean performance of 15,000 runs for each algorithm. Surprisingly, the goodness of performance order was greedy, Lagrange, dual. The greedy and Lagrangian methods only weakly differ in performance– 1% at only a 35% level of significance. The dual algorithm's performance lagged behind the other two. It performed only 73% as well with better than 1% significance.

We attribute the success of the greedy algorithm to the interleaved nature of the

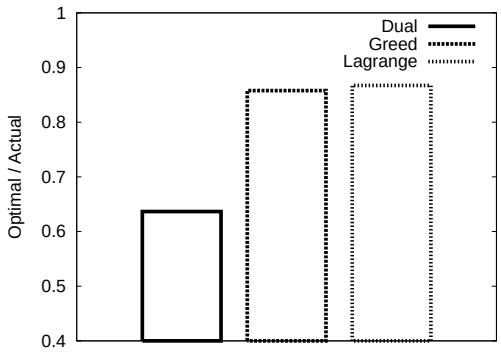

Figure 7.6: Shortest possible divided by mean path length of the three host selection algorithms.

routing and budgeting decisions. Every time that the agent picks a host, it updates its latency estimations and budget.

The dual algorithm performed poorly because it is unable to account for the exact latency incurred in migrating between two hosts. It only uses latency measures on a per-host basis, not between hosts as the other two algorithms do. The oversight may appear to be a drawback, but in environments with high network-volatility transfer times, or when network topology is unknown or dynamic, the ability to route with little inter-host latency information is valuable.

Figure 7.7 plots the latencies of the three algorithms when the agent knew only the mean transfer time from a host to its neighbors, as opposed to the mean transfer time between the host and a specific neighbor in the previous experiment. With restricted knowledge, the dual algorithm outperformed the greedy and Lagrange algorithms by 13% and 4% with better than 1% and 10% levels of significance, respectively.

7.6 Conclusion

We present three algorithms to select hosts for a mobile agent to visit given its set of tasks to complete. The selection problem resembles the NP-complete shortest-constrained-path problem in that the agent chooses a set of hosts to visit to minimize

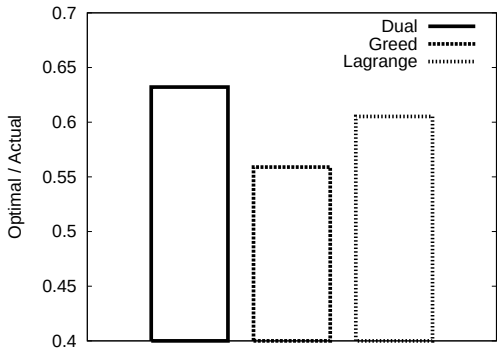

Figure 7.7: Ratio of shortest possible and mean path length of the three host selection algorithms when only execution and mean transfer latencies were known.

its execution time under a budget constraint. Our problem is more difficult, however, in that the cost and latency incurred in a host visit depend on the other hosts in the agent's route.

The relatively poor performance of the two sophisticated algorithms, in comparison to the greedy one, shows that coordination of the routing and budgeting problem is necessary to ensure good performance. Our greedy algorithm updates the agent's budget as it selects every host and outperforms the other two algorithms by 47% and 10%.

Mobile-agent use is promoted for use in wireless and mobile network environments. These settings frequently have volatile or unknown network delays that stem from easily overloaded network links or dynamic network topologies. We consider the impact of uncertainty in network latency on an agent's host selection quality. An agent may not be able to assess network conditions as volatility may be high or the network topology may change over time. Mobile agents are commonly cited solutions to dynamic network problems. It is encouraging to see our dual host-selection algorithm shine under network latency uncertainty. It improves upon the other two algorithms' performance by 10% and 4%.

e_{jk}	The estimated expenditure of the k-th location choice for the j-th task given that the agent visits average hosts for the $l \in [1 \ldots M]/\{j\}$ other tasks.
j	The task index.
K	The number of host choices per task.
k	The host choice index.
τ_{jk}	The estimated latency for the j-th task at the k-th host choice given that the agent visits average hosts for the $l \in [1 \ldots M]/\{j\}$ other tasks.
x_{jk}	The host selection variable.

Table 7.1: Notation used in Chapter 7.

Chapter 8

Risk

Market price fluctuations represent the changes in demand for resources. Price volatility leads to uncertainty in an agent's ability to complete its task at an expected performance level. *Risk* is the uncertainty in an agent's utility. In this chapter, we discuss risk and varying preferences for it; we measure the risk involved in the market example in Chapter 5; we derive a method of forecasting price volatility; and we show how agents and hosts in the example from Chapter 5 can exchange risk.

8.1 Risk

Risk is the uncertainty in an agent's future utility. Certain applications can tolerate more fluctuation than others. There are three categories of preferences towards risk: averseness, neutrality, and inclined. These categories are classified by comparing the expected utility with the utility of the expected outcome.

A risk-neutral agent is indifferent to participation in a gamble with expected value U and the certain outcome that yields utility U. More formally, an agent is risk neutral if it has a utility function, $U(x)$, that for any set of opportunities represented by the random variable X, $E(U(X)) = U(E(X))$. A necessary and sufficient condition for risk neutrality is $\partial^2 U/\partial x^2 = 0$. A real-world example of a risk-neutral agent is an insurance company.

Preference	Expected utility	Second derivative	Example
Averse	$E(U(X)) \leq U(E(X))$	$\partial^2 U/\partial x^2 \leq 0$	home owner
Neutral	$E(U(X)) = U(E(X))$	$\partial^2 U/\partial x^2 = 0$	insurance company
Preferred	$E(U(X)) \geq U(E(X))$	$\partial^2 U/\partial x^2 \geq 0$	desperate gambler

Table 8.1: Conditions and examples that describe risk preferences.

Frequently, an agent can tolerate small changes in utility, but not large ones. For another example, a home owner would be proportionately worse off after a fire destroying an uninsured home than after many years of paying fire insurance without incident. Such agents are risk averse. A risk-averse agent has a utility function, $U(x)$, such that for any set of outcomes denoted by the random variable X, $E(U(X)) \leq U(E(X))$. An identical condition is that $\partial^2 U/\partial x^2 \leq 0$.

An agent is a risk seeker if increasing the agent's wealth by small amounts results in a relatively small gain in utility compared with larger wealth increases. Using the structure that defines the previous risk preferences, a risk-loving agent has a utility function, $U(x)$, such that for the outcomes denoted by the random variable X, $E(U(X)) \geq U(E(X))$. Additionally, $\partial^2 U/\partial x^2 \geq 0$. Table 8.1 summarizes conditions and examples that describe an agent's preference towards risk.

We are now able to present an example of why risk matters to our software agents. Suppose that a host offers an agent the opportunity to compute and that there is uncertainty in the opportunity. The agent may, with known probabilities, compute in time a or d seconds, with expected time b seconds. Figure 8.1 illustrates the gamble. The utility function on the left represents how a risk-neutral agent would view the situation; it is indifferent to computing in b seconds with certainty and taking the gamble between computing in a and d seconds. For a risk-averse agent, whose utility we draw in the right-hand side figure, the indifference point is less favorable; the agent is willing to pay a premium to avoid risk. The difference between the expected utility and the utility of the expected outcome stems from the concavity of the agent's utility function.

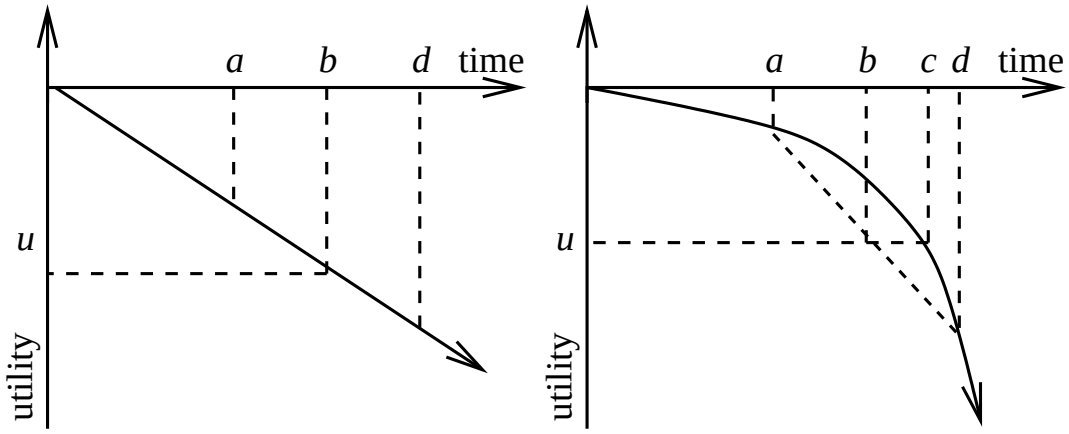

Figure 8.1: The left plot describes a risk-neutral agent's utility function. The agent is indifferent to a certain outcome, b, and a gamble between a and d with expected value b. The plot on the right shows a risk averse utility function. The agent is indifferent to the same gamble in the previous plot and a certain outcome, c.

The known-outcome opportunity to which an agent is indifferent to an uncertain outcome is called the certainty equivalent. That two prospective trading partners have different certainty equivalents allows the two to trade risk. In our illustrated example consider a host that sells computation to an agent with the right-hand side utility function in Figure 8.1. It may be possible for the host to provide computational bounds less attractive than b seconds. If the host can provide the agent with two opportunities, one with uncertainty of computing in either a or d seconds, and the other with certainty in some range of $[b, c]$ seconds, both with equal price, then the agent will choose the latter option. The choice represents an opportunity for the agent to sell its risk to the host.

In this chapter we will assume that agents have a utility function that only depends upon time

$$U = -\psi^{\sum_j \tau_j}, \tag{8.1}$$

where $\psi \geq 1$ is a parameter that describes the agent's preference for stability.

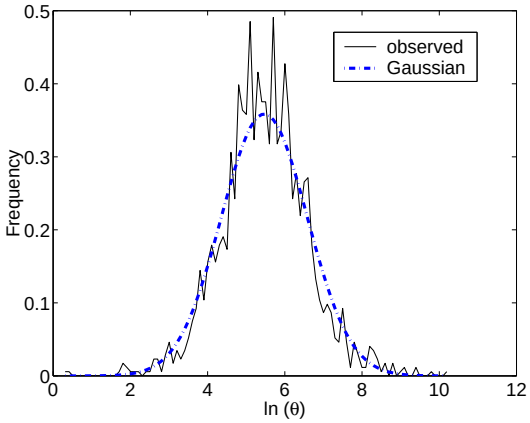

Figure 8.2: A histogram of computation price over the course of an experiment.

8.2 Prediction

In the simulations from Chapter 5, we observed that the price of computation, expressed by the variable θ, fluctuates over the course of an experiment. Figure 8.2 plots a histogram of the price of computation throughout an experiment. The distribution is log-normal.

For some agents, the amount of fluctuation is acceptable. There is enough structure to the volatility that an agent can plan to act when the price falls into a reasonable band. For other agents, the possibility that the price will soar into the long tail of the distribution makes computing unattractive. Knowledge of the structure of the market's volatility can shield an agent from some risk. In this section, we investigate the nature of our market's volatility to derive an appropriate method of risk evaluation.

We begin be adopting a simple model to track the movement of price in our system. Cox, Ross, and Rubinstein [CRR79] developed the binomial options-pricing method to forecast the expected price swings of a security. The binomial model was a response to the newly derived Black-Scholes option-pricing model [BS73]. Black and Scholes use knowledge from physics to describe Brownian motion that predicts the movement of a stock price. The model can be computed in closed form and won them

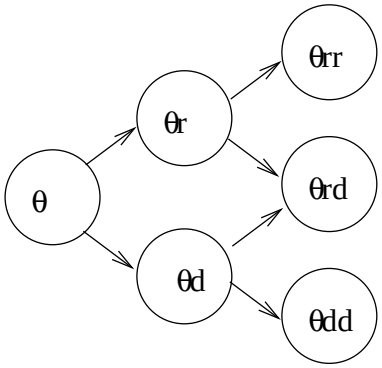

Figure 8.3: A binomial model for predicting the price of a security.

the Nobel Prize in economics, but it relies heavily on assumptions on the empirical nature of the price volatility.

The binomial model discretizes time and assumes that between every time period, the price can move up or down by some relative amount, r or d, respectively. Figure 8.3 graphically shows how price can change in the model through three time steps. The binomial model can be expressed as a Markov model to compute the probability density function of price at any point in the future. In the limit, as the time steps are decreased, the binomial model approximates the Black-Scholes model.

In their basic forms, both the Black-Scholes and binomial models assume that volatility is independent of price. We shall see, though, that the binomial model is easy to augment to track prices when the assumption is dropped. First we examine the properties of price in our model.

Figure 8.4 plots computation price versus the observed volatility conditioned on the price. Figure 8.5 represents a time series of the price of computation at a single host. We note two things. First, the price volatility is strongly related to the price level. Second, the price of computation frequently drops to zero.

We incorporate volatility's dependence on price by observing the volatility at several price brackets. To account for the bracketing, we change the weights and magnitudes of price movement from every node in Figure 8.3 to be conditioned on

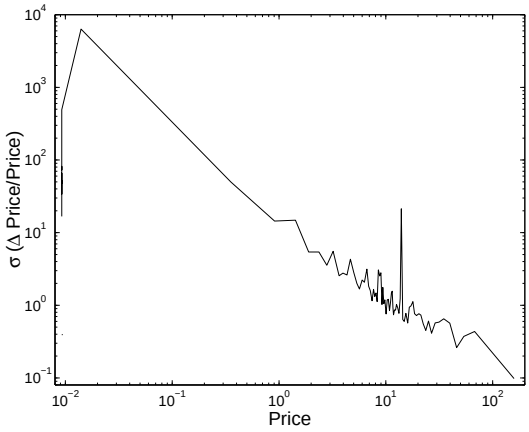

Figure 8.4: Price versus observed volatility at a host in a simulation from Chapter 5.

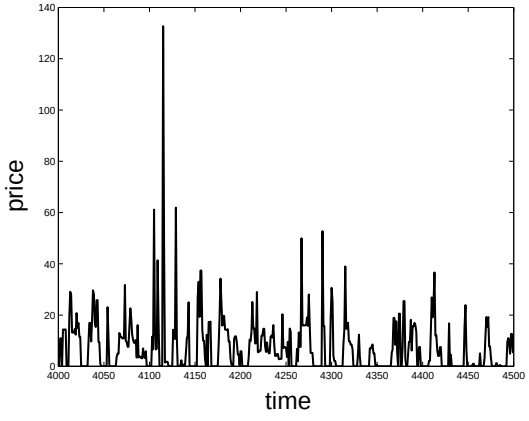

Figure 8.5: Time series of computation price at a host in a simulation from Chapter 5.

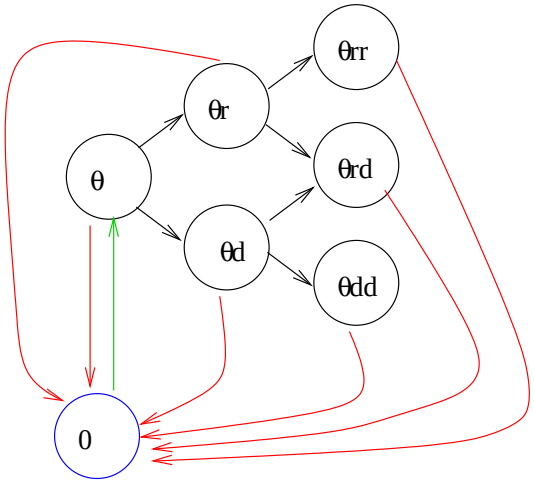

Figure 8.6: The augmented prediction model.

price.

The event where computation price drops to zero reflects that at most one agent consumes computation; there is no congestion. Whether this reflects zero price or just our inability to model price is a philosophical point. We incorporate into the existing binomial model a transition from every state to a special new state that represents the event where the computation price falls to zero. Figure 8.6 represents the augmented binomial model — a Markov state model.

One final issue remains. We must compute the probabilities and magnitudes for each transition in the new Markov model. Cox, Ross, and Rubinstein [CRR79] show that for relative Brownian price movement

$$r := \quad e^{\sigma\sqrt{T/n}} \tag{8.2}$$

$$d := \quad e^{-\sigma\sqrt{T/n}} \tag{8.3}$$

$$p_r := \quad \frac{\mu - d}{r - d}, \tag{8.4}$$

where n is the length of each time period, T is the number of observed time periods, μ is the expected relative change in price, and σ is the standard deviation of the relative

change in price. The parameter r, here, represents the relative amount by which a price will increase; d is the relative amount the price can fall; and p_r is the probability that the price will ascend. In Cox *et al*'s model, the price can only update to one of two values. When the binomial model is applied to highly volatile markets, such as ours, the model can assign p_r a value outside of the range $[0, 1]$ — a meaningless probability.

Another method to compute the probability and magnitudes of price movements is the equal-probability model [Chr97]. It assumes that the security price is as likely to rise as fall, and adjusts the upward and downward movements to correspond to the measured mean and variance. The equal-probability model assigns the binomial probabilities to be

$$r := \frac{2e^{\mu n}}{e^{2\sigma\sqrt{n}}+1} \tag{8.5}$$

$$d := \frac{2e^{\mu n+2\sigma\sqrt{n}}}{e^{2\sigma\sqrt{n}}+1} \tag{8.6}$$

$$p_r := 0.5. \tag{8.7}$$

The equal-probability model is not, however, without problems. When the volatility is high, the model can assign d to be negative and yield a negative price in the following time step — an event that we never observe or has no meaning in our model.

8.2.1 Crash Model

We now can present the method that we use to evaluate risk in our model: the crash pricing model [BKR00]. We extend the standard binomial pricing model to create a more general Markov chain to model θ as the price of computation at a host over time. We label every node in our Markov chain with the pair (θ, t) to represent a possible value for θ during at time t. Each node has a transition to rise in price to node $(r(\theta), t + 1)$ with probability $p_r(\theta)$ and descend in price to node $(d(\theta), t + 1)$

with probability $1 - p_r(\theta) - p_c(\theta)$, where the function $p_c(\theta)$ represents the chance of θ falling to zero in the next time period. Accordingly, each node (θ, t) has a transition to $(0, t+1)$ with probability $p_c(\theta)$.

For θ equal to zero, we compute μ_0 and σ_0^2 to be the mean and variance of the change in θ conditioned on $\theta = 0$ in absolute terms. The computations of μ_0 and σ_0^2 are straightforward from past observations, $p_c(0) = 0$, and $d(0) = 0$. The remaining parameters to compute are $r(0)$ and $p_r(0)$. We start from the definition of the mean and variance,

$$\mu_0 := p_r(0)r(0) \tag{8.8}$$

$$\sigma_0^2 := p_r(0)(\mu_0 - r(0))^2 + p_d(0))\mu_0^2, \tag{8.9}$$

and solve for $r(0)$ and $p_r(0)$ to yield

$$r(0) = \frac{\mu_0^2 + \sigma_0^2}{\mu_0} \tag{8.10}$$

$$p_r(0) = \frac{\mu_0^2}{\sigma_0^2 + \mu_0^2}. \tag{8.11}$$

For θ not equal to zero, we compute $p_r(\theta)$ and $p_c(\theta)$ by looking at the proportion of upward, downward, and extreme downward changes from θ. We also compute the mean and variance of the change in θ conditioned on θ as $\mu(\theta)$ and $\sigma^2(\theta)$, respectively.

From $p_r(\theta),\ p_c(\theta),\ \mu(\theta),\ \sigma^2(\theta)$, and the definitions of mean and variance, we can compute the upward and downward movements of θ. The relative means and variances of the change in θ are

$$\mu(\theta) = p(\theta)r(\theta)/\theta + p_d(\theta)d(\theta)/\theta \tag{8.12}$$

$$\begin{aligned}\sigma^2(\theta) &= p_r(\theta)(\mu(\theta) - r(\theta)/\theta)^2 \\ &+p_d(\theta)(\mu(\theta) - d(\theta)/\theta)^2 \\ &+p_c(\theta)\mu(\theta)^2.\end{aligned} \tag{8.13}$$

We then solve for $r(\theta)$ and $d(\theta)$.

$$r(\theta) = \frac{\theta\mu(\theta)(1 \pm \sqrt{1 + (\frac{p_d(\theta)}{p_r(\theta)} + 1)(1 - \frac{1}{p_d(\theta)} + \frac{\sigma^2(\theta)}{\mu(\theta)^2})})}{p_d(\theta) + p_r(\theta)} \tag{8.14}$$

$$d(\theta) = \frac{\theta\mu(\theta) - p_r(\theta)r(\theta)}{p_d(\theta)} \tag{8.15}$$

It is possible for $d(\theta)$ to be negative, so we take the smallest solution to Equation 8.14 that yields a value greater than $\theta\mu(\theta)$. If $d(\theta)$ is still negative, we assume that all price movement is either upwards or a crash to yield

$$p_r(\theta) = \frac{\mu(\sigma)^2}{\sigma^2(\theta) + \mu(\sigma)^2} \tag{8.16}$$

$$p_c(\theta) = 1 - p_r(\theta) \tag{8.17}$$

$$r(\theta) = \theta\frac{\sigma^2(\theta) + \mu(\sigma)^2}{\mu(\theta)} \tag{8.18}$$

$$d(\theta) = p_d(\theta) = 0. \tag{8.19}$$

This final case assures that $r(\theta)$ is at least $\theta\mu(\theta)$, $p_r(\theta)$ is positive, and $p_c(\theta)$ is non-negative.

We construct a Markov chain using the transition probabilities and magnitudes, $p_r(\theta)$, $p_d(\theta)$, $p_c(\theta)$, $r(\theta)$, and $d(\theta)$. Using the Markov chain, we can estimate, $\text{pdf}(\theta, t, \theta')$, the probability density function for θ t time periods into the future conditioned on its current value, θ'.

To estimate the probability density function, we look through the history of price movements. We separate changes in price uniformly by their starting values and record the changes for all values within the group. From within each group, we

from price bracket \ to price bracket	0	$(0, \theta_1]$	$(\theta_1, \theta_2]$	$(\theta_2, \theta_3]$	$\ldots$
0	$p_c(0)$	0	0	0	
$(0, \theta_1]$	$p_c(\theta_1)$	$p_d(\theta_1)$	$p_r(\theta_1)$	0	
$(\theta_1, \theta_2]$	$p_c(\theta_2)$	$p_d(\theta_2)$	0	$p_r(\theta_2)$	
$(\theta_2, \theta_3]$	$p_c(\theta_3)$	0	$p_d(\theta_3)$	0	
$\vdots$	$\vdots$				

Figure 8.7: The general structure of the matrix that represents the Markov chain in the Crash Model.

assume that prices change to follow the Crash Model and compute the magnitudes and probabilities that the price rises, falls, and crashes from within the group. We generate a matrix that represents the probability of a price transferring from any price group to another. Figure 8.7 shows the structure of the matrix. Most rows have three entries that correspond to the price rising, falling, and crashing. The level to which the price rises or falls depends on the market history.

We calculate the trajectory of a price from the probability that the price will enter any of our groups through composing the price transition matrix with itself. The level of detail that we model prices affects the accuracy of our estimation. Figure 8.8 plots the standard error in price estimation for the actual mean, the sample mean of the previous 1,000 observations, and our crash model trained with 1,000 observations at various levels of estimation granularity for price/volatility conditioning. The degree with which we approximate price movement affects our accuracy. Too much granularity over-fits the history and not enough granularity restricts our ability to model price dynamics. If we choose the right level of granularity, we can reduce the standard error by 11% compared with using the sample mean.

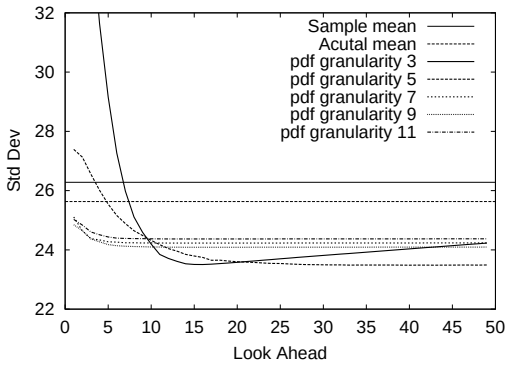

Figure 8.8: Estimation error versus look ahead with several levels of approximation.

8.3 Trading Risk

We would like an attractive computing environment for agents with heterogeneous risk preferences, in spite of the fact that each one operates under the same market fluctuations. In our model, we assume that hosts are risk neutral and that agents are risk averse. The asymmetry in preferences for risk motivates hosts and agents to exchange risk since both parties can benefit from the exchange. In this section, we detail the concept of the asymmetry and show how an agent can determine when there is an asymmetry upon which it can capitalize.

We propose that hosts sell options for computation. A call option is a contract that allows the holder to exercise a purchase at a previously agreed upon price [CW92]. A host can issue a contract that guarantees access to a specified amount of computation at a specified price, the strike price, through the specified time period.

For the host to sell such a contract, it must be able to calculate the probability that the contract will be used. The probability density function, pdf, can tell us the expected loss of revenue at time τ of issuing a call option of with a strike price t. We can then compute the loss of revenue for every period for which the option is valid during $[\tau_{now} + a, \tau_{now} + b]$, where τ_{now} is the current time. If the market for call options is competitive, this loss of revenue, $C(t, x, \tau_{now} + a, \tau_{now} + b, \theta')$, will be equal

to the price of an option with strike price t for cx instructions per second, valid from time a until b, conditioned on the current value of θ equal to θ'.

$$C(t, x, \tau_{now} + a, \tau_{now} + b, \theta') = \sum_{\tau=a}^{b} \int_{\theta=t/x}^{\infty} (\theta x - t) \, \mathrm{pdf}(\theta, \tau, \theta') d\theta \tag{8.20}$$

The quotient θx is the amount that the host could have received for a share of x of its computation in the spot market. The integral is over the space $[t/x, \infty)$ because no rational agent would choose exercise the option if it were cheaper to execute in the spot market.

8.4 Simulation

We now present Algorithm 10, which greedily chooses the next site for the i-th agent, for use with the resource-allocation policy that we use in Chapter 5 to allow agents to purchase computational reservations. Each time an agent chooses its execution location, it evaluates the computational opportunity in the spot market versus buying and exercising a reservation for computation. The agent makes the comparison for every prospective host in the itinerary. The comparisons proceed in order of the task sequence until the agent finds a task for which no attractive reservation contract is offered. The algorithm is greedy in that it always chooses the best immediate decision, but it is also conservative in that it only directs the agent to purchase reservations that can be used with certainty.

We implemented Algorithm 10 inside the same simulation we use in Chapter 5. This time, the agents had the extra ability to reserve computation. Each agent's risk parameter, ψ, was normally distributed and represented the user's preference against risk. We first verified that agents could measure risk inherent in computational markets. Figure 8.9 plots agents' risk parameters versus the observed likelihood that an agent with the risk parameter purchases a computation reservation. We

Algorithm 10 `Greedily Choose Next Site for Agent` i

variable uses	
c	The cost of best execution.
d_k	A pointer to the k-th host with the best spot market.
j	Host choice iterator.
r_k	A pointer to the k-th host from which to reserve computation.
t	A time holder.
u_r	The best guaranteed utility.
u_r	The best utility from the spot market.

```
1: t ← now()
2: for all unreserved tasks k = [1 . . .] do
3:    u_d ← u_r ← −∞
4:    c ← 0
5:    for all hosts j offering service k do
6:       if u_d ≤ E[U(compute at j)] then
7:          u_d ← E[U(compute at j)]          {find best expected spot opportunity.}
8:          d_k ← j
9:          c ← E[cost of computing atj]
10:      end if
11:   end for
12:   for all hosts j offering service k do
13:      if u_r ≤ E[U(compute w/ reservation at j
                 at time t with cost c)] then
14:         u_r ← E[U(compute w/ reservation at j at time t)]
15:         r_k ← reservation j sells          {find best reservation opportunity.}
16:      end if
17:   end for
18:   if u_d < u_r then
19:      break                                 {no more guaranteed computation.}
20:   else
21:      buy r_k
22:      increment t by period of r_k
23:   end if
24: end for
25: if r_1 then
26:   jump to host honoring r_1
27: else
28:   jump to host d_1
29: end if
```

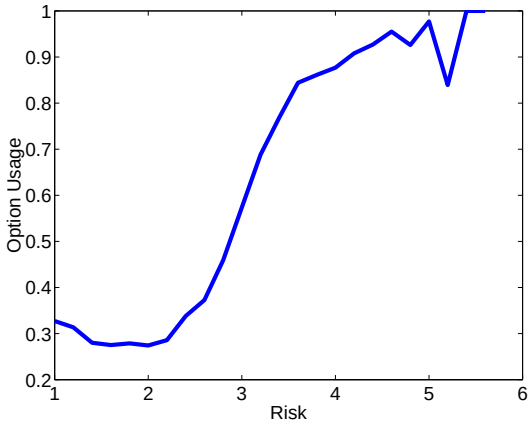

Figure 8.9: Risk versus observed option-usage likelihood.

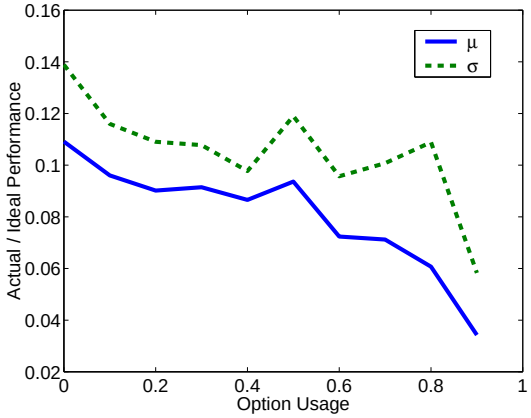

Figure 8.10: Option usage versus performance and performance volatility.

observed a strong relationship between risk averseness and the probability that an agent purchases a reservation. Agents in our simulation could recognize risk.

We next verified that agents using reservations lowered the variance in performance. Figure 8.10 plots the portion of an agent's itinerary guaranteed with reservations versus its expected mean and standard-deviation performance. The variance in an agent's performance certainly decreases after it purchases a reservation. The figure plots the observed deviation in all agents that exercise their reservations. We observe a decrease in deviation, but at a performance cost.

8.5 Conclusions

In this chapter we present a method of measuring the risk involved in computational performance in the simulations from Chapter 5. We show that it is possible to reduce error in congestion estimation by 11% and that a host can use our prediction to evaluate the cost of a reservation or other option for computation.

The application of reservations and options is valuable because the instruments allow agents with heterogeneous risk preferences to use the same market. The ability to trade risk makes our computational markets applicable to more scenarios.

d	The relative amount that the price can fall in the next time step.
p_c	The probability that the price will fall to zero in the next time step.
p_d	The probability that the price will fall by d in the next time step.
p_r	The probability that the price will rise by r in the next time step.
r	The relative amount that the price can rise in the next time step.
ψ	An agent's preference against risk.

Table 8.2: Notation introduced in Section 8.

Chapter 9

Resource Query Implementation

To effectively plan its itinerary, an agent must be able to discover and assess the quality of resources. To further this goal, in this chapter, we describe a system we implemented in Java for agents to query the network for resources, negotiate usage terms, and request a resource. The implementation treats resource description, authentication, allocation, and enforcement as four orthogonal issues. The separation allows a resource owner to balance resource-control requirements and overhead involved in supporting a feature.

9.1 Resources and Lookup

The first issue in resource acquisition is to describe the resource to a lookup service. Our implementation uses Jini [AOS+99] as a lookup service. In Jini, it is possible to locate a network service through specification of an object type with public descriptors used to index an instance of the object. For this purpose, we create a generic class, `Resource`, to represent networked resources available for agent access.

The top level `Resource` class has the descriptors:

- `String name` – The description of the resource, for example, processor.
- `String location` – The physical location of the resource, for example, Sudikoff 120.

- `Double capacity` – The quality of the resource, for example, 400.
- `String metric` – The description of the quality, for example, megahertz.

A principal may subclass `Resource` to add further detail. For example, there may be a subclass `Printer` which is further subclassed `ColorPrinter,` to denote increased capability. An agent that queries for a generic resource will receive entries of more specific types, for example of type `Printer` and `ColorPrinter,` if they are available.

An agent may filter the query by specification of a field value. For example, the agent may wish to find all available resources at a physical location, say Sudikoff room 120. The query that satisfies the request would specify the field "Sudikoff 120" in the `location` field and null, a wild card, in the other fields.

9.2 Authentication

Once the agent locates a resource, it must negotiate with the principal the conditions of access. Frequently, this involves authenticating the agent. There are many methods for an agent to prove its identity with a wide range of security and overhead. Different environments call for different approaches. For this reason, we separate authentication from the rest of the resource query.

When an agent queries the network for a resource, it receives a proxy to the resource that implements the interface `ResourceAccess,` specified in Figure 9.1. In our implementation, the proxy is a reference to a remote object, but it is possible for the proxy to be completely autonomous in its actions. The possibility that each method associated with the proxy can throw a `RemoteException` indicates that the method may have to access the network to perform the agent's request.

The proxy is responsible for authenticating the agent. The agent can request the method of authentication through execution of the proxy's `requiredCredentials` method. The method returns an object that represents the class type, a subclass of the

```
public interface ResourceAccess {
  public ResourceUsage allocated (Credentials c)
  throws InvalidCredentialsException, RemoteException;

  public ResourceUsage allocated (UsagePermit p)
  throws InvalidCredentialsException, RemoteException;

  public Resource getResource ()
  throws RemoteException;

  public void register (Credentials c)
  throws InvalidCredentialsException, RemoteException;

  public UsagePermit request (Credentials c, ResourceUsage r)
  throws InvalidCredentialsException, RemoteException;

  public Class requiredCredentials ()
  throws RemoteException;

  public UsagePermit update (Credentials c, UsagePermit p, ResourceUsage r)
  throws InvalidCredentialsException, RemoteException;

  public void release (UsagePermit p)
  throws InvalidCredentialsException, RemoteException;

  public void relinquish (Credentials c)
  throws InvalidCredentialsException, RemoteException;
} // interface ResourceAccess
```

Figure 9.1: The interface an agent uses to request a resource.

`Credentials` type, that the proxy expects for authentication. The agent examines the type returned by `requiredCredentials` to produce its own `Credentials` instance.

A simple example of a `Credentials` class could contain a clear text string with the agent's name. More sophisticated authentication might include a digitally signed message from a trusted third party with a hash code that verifies that the `Credentials` instance has not been tampered. In either case, the proxy determines a method, which it communicates to the agent using the `requiredCredentials` method.

9.3 Acquisition

The agent gathers the appropriate credentials and registers with the proxy its intent to consume with the `register` method. The agent then may request specific quantities of a resource with the proxy's `request` method, which returns a `UsagePermit` instance that proves the agent has access to the resource. The request need not be fully satisfied, however, and the agent can validate the request through querying the returned `UsagePermit`.

The agent can change its request later with the proxy's `update` method or release a claim with `release`. When the agent plans to no longer access the resource, it can unregister using the `relinquish` method.

9.4 Enforcement

Our interface for resource access does not address enforcement issues. Again, this is an orthogonal issue to other aspects of the framework. Some applications require accurate accounts of each agent's resource usage, wheras some applications only need prescriptive guidelines. Implementation would require coordination of the classes that implement the `ResourceAccess` interface with the operating system or virtual machine.

9.5 Example

Consider an agent that wishes to access a resource. The agent discovers the resource through the lookup service, with which the agent also registers. When the lookup service finds new services, it calls the `discovered` method that the agent defines. We present an example method in Figure 9.2.

The agent specifies to the lookup service that it is interested in services of type `Resource.` In the example, we use `null` as a wild card to match any networked resource. The `ServiceRegistrar lookup` method also takes an integer that specifies the number of matches to return; we are interested in all matches, so we specify the maximum integer value. Once we have found the resources, the agent can catalog them in whatever method it chooses, say in `addResourceProxy.`

Figure 9.3 demonstrates how the agent can use the newfound resource. The general sequence to access a resource is:

1. `register` The agent states its intent to use the resource in the future. In this stage, the initial authentication is performed by the proxy.

2. `request` The agent states a specific amount of the resource it would like to use. The proxy may not fully grant the request and the agent may attempt to change its allocation later using the `update` method.

3. `release` This method is a shorthand for `update` with a `ResourceUsage` argument that specifies zero usage.

4. `relinquish` The agent states that it will have no further need of the resource.

In the context of a mobile-agent application, each of these steps may be performed before the agent migrates to the site. In fact, a host may require the agent to register and state its resource needs before the host will accept the agent.

```
public void discovered (DiscoveryEvent e) {
  ServiceRegistrar[] regs = e.getRegistrars ();
  ServiceMatches matches;
  Class[] sTypes = { ResourceAccess.class };
  ServiceTemplate serviceTemplate = new ServiceTemplate (null, sTypes, null);
  Entry[] entries;
  int i, j, k;

  // Look through all the registrars, for all reported matches.
  // For each match, find the resource description and add the
  // corresponding service, which should be a ResourceAccess instance.
  for (i = regs.length - 1; i >= 0; i--) {
    try {
      matches = regs[i].lookup (serviceTemplate, Integer.MAX_VALUE);

      for (j = matches.totalMatches - 1; j >= 0; j--) {
        entries = matches.items[j].attributeSets;

        for (k = entries.length - 1; k >= 0; k--) {
    // notify the agent that we have found a new resource
          agent.addResourceProxy ((Resource)entries[k],
            (ResourceAccess)matches.items[j].service);
        }
      }
    } catch (RemoteException ex1) {
        System.out.println ("Error looking up template "
        + serviceTemplate + " from registrar " + regs[i]
        + " : " + ex1);
    }
  }
} // discovered
```

Figure 9.2: An example of how an agent discover resources.

```
/** Acquire resource already discovered and stored in knownResources. */
public synchronized void acquireResource (Resource rsc)
throws InvalidCredentialsException {
  ResourceAccess policy =
    (ResourceAccess) knownResources.get (rsc.toString ());
  // for demonstration purposes, attempt to get a lot of resource
  ResourceUsage usage = new ResourceUsage (rsc, Double.MAX_VALUE);

  // determine authentication type
  Class credClass = policy.requiredCredentials ();
  Credentials cred;

  // instantiate credentials
  if (credClass == PriorityCredentials.class)
    // assign an arbitrary priority....
    cred = new PriorityCredentials (new Random().nextInt (17));
  else if (credClass == NullCredentials.class)
    cred = new NullCredentials ();
  else
    throw new InvalidCredentialsException e
      ("Unknown credential type " + credClass
  + " requirement from " + rsc);

  policy.register (cred);
  UsagePermit permit = policy.request (cred, usage);
} // acquireResource
```

Figure 9.3: An example of how an agent can access a known resource.

9.6 Summary

We present a general framework with which resource owners can publish their assets and agents can query. We implemented the framework in Java. Jini provides the lookup service and we handle proxy access through remote method invocation. We separate the issues of lookup, authentication, allocation, and enforcement so that individual resources may be controlled in an independent manner.

Chapter 10

Summary and Future Work

We motivate and present methods that apply markets to computational-resource allocation in distributed systems. In particular, we are interested in how a mobile agent may plan its resource consumption through resource selection and budget planning. We present two models for resource allocation. In one, hosts' and agents' interests are closely aligned, and in the other we make no such assumption. We show that agents can operate effectively in both environments and examine the cost of separation of interests. In both of the environments, we show that an agent's currency holdings determine its performance, though the measurement of success differs in the two models. In the shared-interest model, agents attempt to compute as quickly as possible, but in the separated-interest model an agent balances savings with performance to meet its owner's expectations.

Generally, an agent must complete a series of transactions to complete its itinerary. To the host, these transactions are independent, whereas the agent receives no utility from any particular transaction, but only from the successful completion of all the transactions. The possibility that the agent fails to complete the series adds uncertainty to its utility. An agents can mitigate the uncertainty through exchanging risk with its hosts. This trade is done through the sale of options or reservations for future computation. The result is that volatility is decreased in an agent's utility, at the

expense of reduced performance.

The work in Chapters 5, 6, and 8 optimizes an agent's performance through priority negotiation of a sequence of resources. In Chapter 7 we discuss how an agent can choose the resources that comprise the sequence. When the costs and benefits of selecting each resource is known, our problem is an instance of the shortest-constrained-path problem. Furthermore, any knapsack problem reduces to our resource selection problem. Thus, our problem is NP-hard. We apply an algorithm that solves the shortest-constrained-path problem to select an agent's resources; construct a greedy algorithm; and derive a linear-programming-inspired algorithm that does not need to solve the linear program. In the presence of latency information, the greedy algorithm performs well. We find that our linear-programming inspiration outperforms the other two when the latency information is hidden from the agent and the agent only knows the mean transfer time from one host to all other hosts.

The result is significant because mobile-agent solutions are advertised as effective in environments with unreliable or dynamic network connections. In the context of modern distributed applications, such environments are common.[1]

To show that agents can complete their itineraries in a market environment, we present a framework under which agents can discover and request computational resources in a distributed system. We also note that our model from Chapter 5 has been implemented in D'Agents [CG00]. Prescriptive market-based control or market-based control in an environment of trusted hosts is straightforward. With the addition of a micropayment system such as Millicent [GMA+96] or Netcents [PHS98], an agent's potential to execute can be cryptographically verified and mobile-agent hosts need not trust each other.

In this final chapter of the dissertation, we reflect on the applicability of markets to computational-resource allocation. We look at how we can construct agent's utility

[1]In fact, in the process of writing this section, the author experienced a number of wireless network failures stemming from occasional packet losses.

to solve distributed computational problems. The level of detail that we must model depends on agents' and hosts' interest alignment. Finally, markets can expose the participants to risks that stem from fluctuations in system usage or measurements. We discuss the issues of measurement and risk mitigation.

10.1 Applicability

Markets are not applicable to every situation. They are a general heuristic tool for distributively attacking resource-allocation problems. Few market environments can provide non-trivial guarantees without strong assumptions, but markets can be effective at gathering participants' constraints and finding good solutions with little computational or communication overhead. If a more specific method for optimization is known, it is probably better to use it, in the absence of communication costs.

The strength of market-based approaches lies in decentralized control and incentives for self-interested parties to "cooperate." Market interaction involves a simple language where participants can communicate through their transactions and prices. It is prudent to consider the extent that hosts and agents can communicate with each other and what measures can be taken to prevent or foster communication.

Information exchange is more subtle than the exchange that frequently motivates agents to participate in markets: the desire for trade. Markets can unite parties and enable computation in new scenarios, but the incentives can cloud communication and add uncertainty to an agent's assessment of the world. In Section 10.1.2 we look at the difference in incentives of the models that we presented in Chapters 5 and 6.

10.1.1 Communication

We use the word "cooperation" loosely in the section introduction. The "invisible hand" of economics persuades market participants to exhibit aggregate behavior, such as the uniform consumption distribution over the network or similarly that lower

priority agents should defer until prices, driven by congestion, subside.

More blatant sorts of cooperation can benefit some agents at others' expense. For example, a seller should generally discourage its buyers from forming a cartel where buyers conspire against the seller to compete less aggressively for a commodity. Curiously, it is possible that system designers have agents' best interest in mind in communication prevention; Schaerf *et al.* show instances where communication between agents causes their performance to suffer from conservative resource consumption and under-utilized system resources [SST95].

One method to prevent cartel formation is to limit information transfer between agents. A mobile-agent host may decide not to support a yellow pages that allows an agent to locate all other agents at the site. Such a barrier may hinder functionality, however; at some point, such information may be critical to observing and reacting to the environment. Recall the naive-learning example in Section 6.5.1. Without a history of transactions, it is difficult for an agent to learn the range of competitive bids. A seller can publish previously cleared transactions to give agents a better understanding of the market.

Devious agents can use these transaction publications as signals to collude, however. The slowly falling price of a commodity can signal agents to bid conservatively for the commodity. The seller can encourage competitive behavior by not trading at the lower price, but the inaction may be expensive if the seller already accumulates rent.

In our work, we have dealt almost exclusively with auctions for price discovery. The field of computational-resource trade is still young. It is difficult to determine the value and consumption patterns of users. Furthermore, it is hard to identify the market for computation.

Mobile agents have many uses. One user may simply wish to distribute the computational load across the network. Another user may wish to relocate computation

close to a data source, which may be duplicated at several sites around the network. We have enough examples now to see that a market for computation can resemble a monopoly, oligopoly, or perfect competition depending on an agent's ability to substitute computation at one host for another. Furthermore, a host may be oblivious to agents' abilities to substitute its computation.

Our work leverages auctions and allows each agent to decide in what sort of market to operate. The choice of auctions over fixed-price policies may not matter. Goldberg *et al.* show that in many cases the difference in revenue generated from the use of auctions and set prices in the sale of zero marginal-cost goods is a fixed constant factor [GHW01].

10.1.2 Incentives

Currency exchange for resource access enables many applications. For example, a user passing through an airport may wish to access a news database. The user would like to launch an agent to visit the database, filter it for interesting stories, and return the results to the user's palm-top computer for perusal once the user has departed.

From the database owner's perspective, there is no reason to entertain the user's request unless the user can show that the work benefits the database owner. Our proposition that the user pay the database owner for computation and data access ensures that both sides benefit from use.

Traditional business models rely on advertisements to pay for data access. There are few Internet companies that have profited under advertising-based business models. Internet users are already overwhelmed with information and may not take heed to yet another advertisement. Additionally, there are many technologies to filter advertisements from a user's view.

Our market-based allocation charges a user directly for the resources she uses. Furthermore, the service provided to the user is flexible. The user can tailor her

computation needs or contract out computation to agents in the network.

An issue for the user (or her agents) to consider in resource allocation is resource ownership. In Chapter 5 we look at a scenario where users share ownership of resources, whereas Chapter 6 looks at a more general model. We call the former a closed-interest market, since all agents and resources operate under a common administrative domain. The latter model is an open-interest market and does not make any assumptions that concern ownership or identity.

The open-interest model can be applied to more situations, and is the easier of the two computation models to bring to market. The separation of ownership and consumption utility is costly, however. The division presents a barrier for information exchange that results in less efficient allocation. Ideally a resource should be used by the party that values it most, but without communication between agents, it is impossible to determine who values the resource most.

A host that does not sell all of its computation is not by itself a sign of inefficiency. The model assumes that each host places a value on computation for private consumption. Unsold computation in most scenarios still has a value under the model.

An additional factor to consider is that agents in the open-interest model are driven to meet a performance expectation. There may be relatively little utility to be gained from executing faster than expectation. Therefore, an agent would be interested primarily in a particular quantity of computation. The benefit of extra priority will increase the agent's chance of completion faster than expectation, but we observe in simulations that an agent's belief of a bid's acceptance is sharp. That is, the perceived difference in utility between bids rejected and bids accepted by the host is narrow. That agents do not receive larger allocations does not by itself imply inefficiency. An agent may not believe that it benefits significantly from faster computation.

10.2 Utility Model

In the previous section, we note that the choice of utility functions effects resource allocation. In closed markets, an agent's strategy can focus on maximization of a performance metric. In open markets, in addition to performance, an agent is concerned with the value of savings, since the agent gives up its potential to compute to the host. The host is concerned with revenue since it provides service to agents with arbitrary goals.

Under the closed-interest market, we can directly prioritize an agent by its endowment relative to the amount of work expected from it. The problem of prioritization becomes more complicated in the open model. We adopt a quasi-linear utility function to model the value of savings. The utility family is too simple to express priority solely through endowment, however. With quasi-linear utility, there is no wealth effect. Thus, if an agent has quasi-linear utility, the agent has no rationale to compute more quickly with a larger endowment.

This motivates us to look at other qualities that contribute to utility. In Chapter 6 we look at utility derived from an agent's ability to meet a deadline weighted with the cost of the computation. With proper calculation of the parameters, we can roughly express the notion that wealthier agents should compute more quickly than poorer ones. The user can express her desire for high-priority computation through imbuing an agent with a large endowment *and* a short expectation for completion. The model has a shortcoming in that the agent strives to obtain a specific performance expectation, or fails; the agent cannot recognize compromise.

It would be nice to alleviate the user of some of the parameter computation. Our future research will consider utility functions that more directly model user satisfaction. We have had mixed success in other utility models. We experimented with Cobb-Douglas utility [BKR98b], another simple model, but it was difficult to compute

proper weights for consumption and savings. In extending the use of Cobb-Douglas utility to model multi-hop itineraries, we had difficulty modeling the value of savings to guarantee good performance.

Another complication is repeated interaction of agents and hosts. There is a difference in the theoretical optimal play of a non-repeated game and what is observed in everyday interactions. Consider, for example, the participation in second-price auctions on eBay. Game theory asserts that a bidder should submit her evaluation for a good. If this were the case, then one would expect the distribution of bids in an auction to be uniform over time, or weighted towards the early part of an auction if bidders always actively search for an auction. In practice, though, more than half of bids are submitted within the last 10% of the auction's duration [BH01]. Bajari and Hortacsu conclude that bidders worry about inciting interest and competition in the auction, though they also conclude that the revenue generated from eBay auctions is not significantly different from book value in their study of rare coin auctions.

A complication with analysis of eBay auctions is that the bidder's identity is not completely secret. Every bid has an associated electronic mail address. It is imaginable that one's bidding strategy could change because of repeated interactions with known identities. Economists recognize the difficulty and, as such, the literature base on repeated-mechanism design is small. A common solution to the problem is to assume that one party using the mechanism has unlimited patience, and a trivial solution ensues [KL98].

10.3 Reservation-Based Access

Patience is not the only temporal issue in utility. An agent may wish to have the right to access a resource over a time period, even though it will not require access through much of the period. In Chapter 8 we examined the possibility that agents purchase options to compute. Primarily we were interested in options as a means of

measuring the value of a reservation to compute, but more general options are useful, too. Many network applications require intermittent network access to report events, which require processing at another site.

Such an application would value a resource, even when the application does not access the resource. The aspect of price measurement through congestion is interesting in this light. In Section 6.2.3 we use congestion to measure the price of computation. When there is no resource usage, we cannot calculate a price, so assume it to be zero. That does not necessarily correlate with value, however, when there are applications that value the opportunity for access. Future research will more accurately measure the value of computation to directly include the opportunity of access.

We measure the virtues and costs of demand versus reservation-based resource allocation in Chapter 8. Our experiments quantify our intuition: that reservations have an efficiency cost. It is useful to look at the spectrum of reservation versus demand-based resource allocation. The applicability of any approach on the spectrum depends on the application environment. The resource-access option is a powerful instrument that allows agents to choose where along the spectrum to execute. An agent can choose to ignore reservation-based access or embrace it wholeheartedly by choosing to purchase resource access exclusively through front-loaded options. An agent with intermediate risk preferences can evaluate resource-access options and decide when the market for reservation-based favors the agent. Ideally, we would like to cater to intermediate preferences through establishment of more general types of options that we have not yet explored, such as options with a positive strike price.

10.4 Future Directions

Preceding sections describe incremental improvements in our research. We conclude the dissertation in this section by fleshing out more substantial improvements as future research: hierarchical resource-access planning, system robustness, and further

analysis of repeated interaction in resource allocation.

10.4.1 Interface

An application that leverages market-based control assesses the user's preferences. Ideally, the user should divulge as little information as possible. One could imagine an application querying the user for her overall level of satisfaction and that the agent can infer an allocation of the user's resources to improve the level of satisfaction.

The user should not have to know the exact parameters that agents use to optimize performance. In the context of the resource allocation that we describe in Chapter 5 where an agent cannot transfer its savings back to its owner, the user must be able to determine how much currency with which to endow an agent. This decision is a meta resource-allocation problem.

A more general system involves launching agents that participate in different resource-allocation policies. The user, or a foreman agent, must determine how to allocate currency and tasks to other agents. This allocation can be repeated to form a hierarchy of resource-allocation problems. At the top level, the user expresses her pleasure with current system performance and her potential for future resource consumption.

10.4.2 System Resilience

Consider system robustness. In Chapter 2 we argue that a motivation for market use in distributed-system resource allocation is that each agent's budget constraint enforces a degree of fault tolerance on the system. It will be useful to look both empirically and analytically at a malicious agent's ability to conduct a denial-of-service attack.

In Section 8.2, we observe a distribution of the rate at which a host collects income for its computation. From the histogram, we can compute the mean value of

computation and derive a lower bound on the price of eliminating all other agents access to the market. To derive the price of a complete denial-of-service attack, we can look at the distribution of the maximum rate that any agent will pay for computation, though in a real-world application, a host may not wish to publish that information, or the publication could affect an agent's optimal bidding strategy.

A complete denial-of-service attack may not be necessary, however. Additionally, an agent may be able to change its bidding behavior to mitigate some of the delay conferred by a denial-of-service attack. Therefore, we propose further research to analyze existing systems and build new systems to repel such attacks.

10.4.3 Repeated Interaction

Traditional mechanism design considers how an agent participates in a mechanism isolated from its decisions in past or future outcomes with mechanism participation. It is a common assumption that the agent has beliefs of the distributions of its competitors' preferences, likely generated through past interactions. The field ignores the fact that other agents might learn from the agent's play as a result of outcome of the mechanism, however.

The problem is not trivial. Researchers in artificial intelligence often apply heavyweight dynamic-programming approaches to reason about the interactions of one encounter with the environment to the next. The computation becomes more difficult when one considers how the environment adapts to the agent's actions.

Like much of the work in microeconomics, our work assumes that an individual software agent cannot effect the rest of the market in a major way. We have dealt with repeated interaction through optimization of a simple game with a finite number of rounds. It will be interesting to consider the effect of adding uncertainty to the number of rounds, generalizing agents' utilities, or using other mechanisms.

10.5 Conclusion

We research markets that regulate distributed computational systems. The motivation for the application of markets is twofold. Economics provides a rich tool set for analysis of distributed allocation problems. In particular, economic analysis provides insight on planning problems as well as methods for distributively making resource-consumption decisions. Secondly, a market where participants securely exchange valuable currency enables new distributed applications where users do not trust each other.

We focus on mobile-agent systems because they provide the mechanisms for agents to exchange computational resources. One agent can negotiate with another access to communication, data access, or computation. While computation becomes cheaper and cheaper everyday, it is not always possible to locate computation near the user, nor can a user always rely on her own resources to solve a problem at hand.

For example, a traveler in an airport may wish to filter news stories based upon previous, perhaps private, experience. Rather than rely on a static, application specific framework, the user can tap into the airport's network of computers and inject generic computation into the network.

Another example is a research institution with sporadic needs for high performance computation. Rather than purchase and support expensive machinery, users at the institution can dynamically lease computation from external sources. This pattern of resource access shows how a system administrator can inexpensively maintain a flexible domain.

Limited market-based computational-resource allocation has already been adopted by several companies. Companies such as United Devices[2] and XDegrees[3] broker computation between personal-computer owners and researchers. Enron Communica-

[2] `http://www.uniteddevices.com`
[3] `http://www.xdegrees.com`

tions[4] allows its customers to trade network-bandwidth access. Mojo Nation[5] governs access of distributed resources with a currency model. Each of these examples shows how markets empower network applications.

So far, the exchange in these markets has been one dimensional: one transaction does not effect another. In this dissertation, we consider expanding resource planning over sequences of related tasks. We consider an agent's ability to select resources as well and negotiate the priority at which the agent consumes the resource. We show how the process can significantly improve application performance over traditional resource allocation.

Within the systems that we study, an agent can assess the volatility of its expected performance to choose resources and negotiate reserved resource access. We propose mechanisms that allow an agent to consider three dimensions of resource quality: cost, performance, and risk. The agent's instruments for execution of its choice are location, time, and mechanism choice (demand versus reservation-based).

We synthesize and study examples of markets with differing ownership models to illustrate efficiency and information losses. In Chapter 5, we propose a model where users collectively own resources. The shared-resource scenario motivates an agent to disclose its needs and the system is able to efficiently allocate resources.

In other situations, it may not be feasible to motivate such disclosure. The information may not be known by the agent, collection may be too expensive, or the nature of the application may distribute resources over several domains. We relax the ownership and agent-utility homogeneity assumptions in Chapter 6 and show how an agent can plan its resource consumption with weakened trust in its hosts under congestion uncertainty.

This dissertation shows the value of economic thought in distributed-system design and optimization. We believe it to be a valuable tool in creating new environments

[4] `http://www.enron.com`
[5] `http://www.mojonation.net`

for computation as well as for smooth regulation of existing applications.

Appendix A

Notation

$D[f(n)]$	The derivative of the discrete function $f(n)$.
E_i	The i-th agent's endowment.
K	The number of host choices per job.
M_i	The number of jobs that the i-th agent must perform.
N_{ij}	The number of agents at the j-th host that the i-th agent visits.
U_0	The host's utility.
U_i	The i-th agent's utility.
Δ	The length of time for which the host guarantees computation to auction winners.
α_i	A parameter that describes the i-th agent's bid function. $\alpha_i := E_i - \sum_{j\neq 1}^{M_i} q_{ij}\theta_j^{-i}/c_{ij}$
β_i	A parameter that describes the i-th agent's bid function. $\beta_i := q_{i1}/c_{i1}$
ϵ	The approximation gap that the host uses for bid selection.
γ_i	A parameter that describes the i-th agent's bid function. $\gamma_i := \max\{\epsilon, \sum_{j\neq 1}^{M_i} q_{ij}\sqrt{\theta_j^{-i}}/c_{ij}\}$
$\hat{\tau}_i$	The latency that the owner of the i-th agent expects the agent to incur.
κ_i	The precision of a user's expectation that the i-th agent completes in time less than $\hat{\tau}_i$.
ψ	An agent's preference against risk.
$\rho(x)$	The density-maximized bid of size x.
τ_{ij}	The time the i-th agent spends at the j-th host.
τ_{jk}	The estimated latency for the j-th job at the k-th host choice given that the agent visits average hosts for the $l \in [1 \ldots M]/\{j\}$ other jobs.
θ_j^{-i}	The sum of all bids, except the i-th agent's, at the j-th host.
θ_j	The sum of all bids at the j-th host.

a	The number of auctions that the agent must win to complete its current job at the host.
b	An approximate upper bound on the number of auctions in which an agent can participate and receive positive utility.
c_{ij}	The capacity in computational units per second that the j-th host visited by the i-th agent can sustain.
d	The relative amount that the price can fall in the next time step.
e_{jk}	The estimated expenditure of the k-th location choice for the j-th job given that the agent visits average hosts for the $l \in [1 \ldots M]/\{j\}$ other jobs.
$f_{ij}()$	The i-th agent's bid given all other agents' bids, defined byEquation 5.10.
	The rate at which the i-th agent pays the host conditioned on the rate at which the host collects from all agents excluding the i-th.
$g_{ij}()$	The i-th agent's bidding function at the j-th host, definedby Equation 5.17.
	The rate at which the i-th agent pays the host conditioned on the rate at which the host collects from all agents.
$h(x)$	Shorthand for $\exp(\kappa_i \sum_{j=1}^{M_i} q_{ij}/c_{ij}x_{ij} - \hat{\tau}_i)$.
i	The agent index.
j	The job index.
k	The host choice index.
$p(x,t)$	The believed probability that a bid for amount x at price t will be accepted by the host.
p_c	The probability that the price will fall to zero in the next time step.
p_d	The probability that the price will fall by d in the next time step.
p_r	The probability that the price will rise by r in the next time step.
$q(x,t)$	The complement of $p(x,t)$.
q_{ij}	The size of the i-th agent's j-th job.
r	The relative amount that the price can rise in the next time step.
s	The scaling factor used by the host to pack the big bids.
t_{ij}	The monetary transfer from the agent to the host for computation of the i-th agent's j-th job.
u_{ij}	The rate at which the i-th agent pays the j-th host.
$v*$	The host's reserve price for computation.
v_0	The host's marginal valuation of the resource.
x_{ij}	The fraction of capacity for the j-th job that the i-th agent receives.
x_{jk}	The host selection variable.

Bibliography

[AAB98] Yair Amir, Baruch Awerbuch, and R. Sean Borgstrom. A cost-benefit framework for online management of a metacomputing system. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 140–147, Charleston, SC, October 1998. ACM Press.

[AKNS98] A. Anastasiadi, S. Kapidakis, C. Nikolaou, and J. Sairamesh. A computational economy for dynamic load balancing and data replication. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 166–180, Charleston, SC, October 1998. ACM Press.

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

[AOS+99] Ken Arnold, Bryan O'Sullivan, Robert W. Scheifler, Jim Waldo, and Ann Wollrath. *The Jini Specification*. Addison-Wesley, Boston, MA, 1999.

[ASGH95] David Abramson, Rok Sosic, Jonathan Giddy, and B. Hall. Nimrod: a tool for performing parametised simulations using distributed workstations. In *Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing*, pages 112–121, Virginia, August 1995.

[BAG00] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the Fourth International Conference on High Performance Computing in Asia-Pacific Region*, pages 283–289, Beijing, China, 2000.

[BGS99] Craig Boutilier, Moises Goldszmidt, and Bikash Sabata. Sequential auctions for the allocation of resources with complementarities. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 527–523, Stockholm, Sweden, 1999.

[BH01] Patrick Bajari and Ali Hortacsu. Winner's curse, reserve prices and endogenous entry: Empirical insights from eBay auctions, January 2001. Stanford Economics Working Paper, `http://www.stanford.edu/~bajari/wp/auction/ebay.pdf`.

[BKR98a] 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.

[BKR98b] Jonathan Bredin, David Kotz, and Daniela Rus. Utility driven mobile-agent scheduling. Technical Report PCS-TR98-331, Dartmouth College, May 1998.

[BKR99] Jonathan Bredin, David Kotz, and Daniela Rus. Economic markets as a means of open mobile-agent systems. In *Proceedings of the Workshop "Mobile Agents in the Context of Competition and Cooperation (MAC3)" at Autonomous Agents '99*, pages 43–49, May 1999.

[BKR00] Jonathan Bredin, David Kotz, and Daniela Rus. Trading risk in mobile-agent computational markets, July 2000. Presented at the Conference for Computation in Economics and Finance, Barcelona, Spain.

[BMI+00] Jonathan Bredin, Rajiv T. Maheswaran, Cagri Imer, Tamer Başar, David Kotz, and Daniela Rus. A game-theoretic formulation of multi-agent resource allocation. In *Proceedings of the Fourth International Conference on Autonomous Agents*, Barcelona, June 2000.

[Boa83] Mary L. Boas. *Mathematical Methods in the Physical Sciences*. John Wiley & Sons, New York, NY, 1983.

[BPR98] Mario Baldi, Gian Pietro Picco, and Fulvio Risso. Designing a videoconference system for active networks. In *Proceedings of the Second International Workshop, Mobile Agents '98*, pages 273–284, Stuttgart, Germany, September 1998.

[BPS95] Dimitri P. Bertsekas, Stefano Pallottino, and Maria Grazia Scutella. Polynomial auction algorithms for shortest paths. *Computational Optimization and Applications*, 4:99–125, 1995.

[BPW98] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile agents for network management. *IEEE Communications Surveys*, September 1998.

[BS73] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[BTS+98] Godmar Back, Patrick Tullmann, Leigh Stoller, Wilson C. Hsieh, and Jay Lepreau. Technical report UUCS-98-015, Java operating systems: Design and implementation. Technical report, University of Utah, August 1998.

[CCDS96] Scott H. Clearwater, Rick Costanza, Mike Dixon, and Brain Schroeder. Saving energy using market-based control. In Clearwater [Cle96], pages 253–273.

[CG00] Ezra E. K. Cooper and Robert S. Gray. An economic CPU-time market for D'Agents. Technical Report TR2000-375, Dartmouth College, June 2000. Undergraduate honors thesis. Advisor: Bob Gray.

[Chr97] Neil A. Chriss. *Black-Scholes and Beyond Option Pricing Models*. McGraw-Hill, New York, NY, 1997.

[Chv83] Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, NY, 1983.

[Cla71] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[Cle96] Scott H. Clearwater, editor. *Market-Based Control*. World Scientific, Singapore, 1996.

[CMM97] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: A multiagent system for distributed resource allocation. In *Proceedings of the First International Conference on Autonomous Agents*, Marina Del Ray, CA, 1997. ACM Press.

[CRR79] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7:229–263, 1979.

[CvE98] Grzegorz Czajkowski and Thorsten von Eicken. JRes: a resource accounting interface for Java. In *Proceedings of the 1998 ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Vancouver, BC, October 1998.

[CW92] Thomas E. Copeland and J. Fred Weston. *Financial Theory and Corporate Policy*. Addison-Wesley, Reading, MA, third edition, 1992.

[CW98] John Q. Cheng and Michael P. Wellman. The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Journal of Computational Economics*, 12:1–23, 1998.

[CZ96] Ken Calvert and Ellen Zegura. GT-ITM: Georgia Tech internetwork topology models, 1996. http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz.

[DM88] K. Eric Drexler and Mark S. Miller. Incentive engineering for computational resource management. In Huberman [Hub88], pages 231–266.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In E. Brickell, editor, *Advances in Cryptology– CRYPTO 92 Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer-Verlag, 1992.

[ELZ86] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[EWCS96] Yasuhiro Endo, Zheng Wang, Bradley Chen, and Margo Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the 1996 Symposium on Operating System Design and Implementation*, Seattle, WA, October 1996.

[FL98] Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, MA, 1998.

[FLBS99] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[FNSY96] Donald F. Ferguson, Christos Nikolaou, Jakka Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In Clearwater [Cle96], pages 156–183.

[FT96] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1996.

[FYN88] Donald F. Ferguson, Yechiam Yemini, and Christos Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. In *Proceedings of the International Conference on Distributed Systems*, pages 491–499, 1988.

[GD98] Steven Gjerstad and John Dickhaut. Price formation in double auctions. *Games and Economic Behavior*, 22(1):1–29, January 1998.

[GHW01] Andrew V. Goldberg, Jason D. Hartline, and Andrew Wright. Competitive auctions and digital goods. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington, DC, January 2001.

[Gib73] Allan Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41(4):587–601, July 1973.

[Git89] John C. Gittins. *Multi-armed Bandit Allocation Indices*. John Wiley and Sons, Chichester, Great Britain, 1989.

[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.

[GKP94] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, MA, second edition, 1994.

[GL77] Jerry Green and Jean-Jacques Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45(2):427–438, March 1977.

[GMA+96] Steve Glassman, Mark Manasse, Martín Abadi, Paul Gauthier, and Patrick Sobalvarro. The Millicent protocol for inexpensive electronic commerce. *World Wide Web Journal*, 1(1), Winter 1996. Also in Fourth International World Wide Web Conference, December 1995.

[Gra97] Robert Gray. *Agent Tcl: A flexible and secure mobile-agent system*. PhD thesis, Dartmouth College, June 1997. Available as Dartmouth Computer Science Technical Report TR98-327.

[Gro73] Theodore Groves. Incentives in teams. *Econometrica*, 41(3):617–631, July 1973.

[GSW97a] Alok Gupta, Dale O. Stahl, and Andrew B. Whinston. Priority pricing of integrated services networks. In Lee W. McKnight and Joseph P. Bailey, editors, *Internet Economics*, pages 323–352. MIT Press, Cambridge, MA, 1997.

[GSW97b] 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.

[GW97] Michel X. Goemans and David P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Hochbaum [Hoc97a], chapter 4, pages 144–191.

[HC96] Kieran Harty and David R. Cheriton. A market approach to operating system memory allocation. In Clearwater [Cle96], chapter 6, pages 126–155.

[Hen99] John Hennessy. The future of systems research. *IEEE Computer*, 32(8):27–33, August 1999.

[HL89] David W. Jr. Hosmer and Stanley Lemeshow. *Applied Logistic Regression*. Wiley, New York, NY, 1989.

[Hoc97a] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA, 1997.

[Hoc97b] Dorit S. Hochbaum. Various notions of approximations: Good, better, best, and more. In *Approximation Algorithms for NP-Hard Problems* [Hoc97a], chapter 9, pages 346–398.

[Hub88] Bernardo Huberman, editor. *The Ecology of Computation*. Elsevier Science Publishers/North-Holland, New York, NY, 1988.

[IK75] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, 22(4):463–468, October 1975.

[Joh98] Dag Johansen. Mobile agent applicability. In *Proceedings of the Second International Workshop, Mobile Agents '98*, pages 80–98, Stuttgart, Germany, 1998.

[Joh99] Claudia Johnson. Enron Communications announces first commodity bandwidth trade, December 1999. Press release `http://www4.enron.com/corp/pr/releases/1999/ene/bandwidth.html`, last visited March 2001.

[KG99] David Kotz and Robert S. Gray. Mobile agents and the future of the Internet. *ACM Operating Systems Review*, 33(3):7–13, August 1999.

[KKO98] Mehmet Karaul, Yannis A. Korilis, and Ariel Orda. A market-based architecture for management of geographically dispersed, replicated web servers. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 158–165, Charleston, SC, October 1998. ACM Press.

[KL98] Ehud Kalai and John O. Ledyard. Repeated implementation. *Journal of Economic Theory*, 83(2):380–317, December 1998.

[KS89] James F. Kurose and Rahul Simha. A microeconomic approach to decentralized resource sharing in distributed systems. *IEEE Transactions on Computers*, 38(5):705–717, 1989.

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

[Law77] Eugene L. Lawler. Fast approximation schemes for knapsack problems. In *Proceedings of the Eighteenth Annual Symposium on Foundations of Computer Science*, pages 206–213, Providence, Rhode Island, November 1977. IEEE Computer Society.

[LBDL99] Chris Langton, Roger Burkhart, Marcus Daniels, and Alex Lancaster. The Swarm simulation system, 1999. http://www.santafe.edu/projects/swarm.

[LBG+] Jeff Lawson, Adam L. Beberg, Peter Gildea, David McNett, Chris Chiapusio, Peter DeNitto, and Tim Charron. distributed.net. `http://www.distributed.net,` visited March 2000.

[LO86] Will E. Leland and Teunis J. Ott. Load-balancing heuristics and process behavior. In *Proceedings of the Performance '86 and ACM Sigmetrics 1986 Joint Conference on Computer Performance Modeling, Measurement and Evaluation*, pages 54–69, 1986.

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

[LO99] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, March 1999.

[Lou99] Franois-Xavier Le Louarn. Jum, a Java usage monitor, 1999. `http://www.iro.umontreal.ca/~lelouarn/jum.html`.

[McIB99] Rajiv T. Maheswaran, Çagri Imer, and Tamer Başar. Agent mobility under price incentives. In *Proceedings of 38th IEEE Conference on Decision and Control*, Phoenix, AZ, December 1999.

[MCWG95] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, New York, NY, 1995.

[Miy] Shichirou Miyashita. The e-Marketplace. `http://www.trl.ibm.co.jp/aglets/emplace/emplace.html,` visited April 1999.

[MKM99] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes. Cooperating mobile agents for dynamic network routing. In Alex Hayzelden, editor, *Software Agents for Future Communications Systems*, chapter 12. Springer-Verlag, New York, NY, 1999.

[MMV95] Jeffrey K. MacKie-Mason and Hal R. Varian. Pricing the Internet. In Brian Kahin and James Keller, editors, *Public Access to the Internet*, pages 269–314. MIT Press, Cambridge, MA, 1995.

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

[Mol01] David Molnar. Resource allocation, proofs of work, and consequences. *ACM Crossroads Student Magazine*, March 2001. `http://www.acm.org/crossroads/columns/onpatrol/march2001.html`.

[MPT00] Ajay Mohindra, Apratim Purakayashta, and Prasannaa Thati. Exploiting non-determinism for reliability of mobile agent systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, New York, NY, June 2000.

[MR97] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, 1997.

[MS83] Roger B. Myerson and Mark A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 28:265–281, 1983.

[Mul98] Tomasz Muldner. Mobile computing at Acadia University. Dartmouth College Computer Science Colloquium, 1998. Slides at http://evilqueen.acadiau.ca/presentations/mobileagents.ppt.

[NFS00] Dushyanth Narayanan, Jason Flinn, and Mahadev Satyanarayanan. Using history to improve mobile application adaptation. In *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications*, pages 31–40, San Diego, CA, December 2000.

[Nis99a] Noam Nisan. Algorithmic mechanism design. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, Atlanta, Georgia, May 1999.

[Nis99b] Noam Nisan. Algorithms for selfish agents – mechanism design for distributed computation. In *Proceedings of the Symposium on Theoretical Aspects in Computer Science*, Trier, Germany, March 1999.

[ns01] The network simulator – ns2, 2001. http://www.isi.edu/nsnam/ns.

[OR96] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* MIT Press, Cambridge, MA, 1996.

[Par99] David Parkes. iBundle: an efficient ascending price bundle auction. In *Proceedings of the ACM Conference on Electronic Commerce*, Denver, CO, November 1999.

[PHS98] Tomi Poutanene, Heather Hinton, and Michael Stumm. NetCents: A lightweight protocol for secure micropayments. In *USENIX Workshop on Electronic Commerce*, pages 25–36. USENIX Association, September 1998.

[Pis95] David Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, February 1995.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Dover Publications, Mineola, New York, 1998.

[PT98] Ioannis Ch. Paschalidis and John N. Tsitsiklis. Congestion-dependent pricing of network services. Technical report, Boston University, 1998.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 1992.

[RN98] Ori Regev and Noam Nisan. The POPCORN market— an online market for computational resources. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 148–157, Charleston, SC, October 1998. ACM Press.

[RPM+99] Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, and Neil Stratford. Xenoservers: Accountable execution of untrusted programs. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*. IEEE Computer Society, March 1999.

[San99] Tuomas Sandholm. An algorithm for optimal determination in combinatorial auctions. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[Sat75] Mark A. Satterthwaite. Stategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.

[Sch98] John M. Schnizlein. How can routers help Internet economics? In *Proceedings of the First International Conference on Information and Computation Economies*, pages 52–59, Charleston, SC, October 1998. ACM Press.

[SDK+94] Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, and Carl Staelin. An economic paradigm for query processing and data migration in Mariposa. Technical report, University of California, Berkeley, April 1994.

[SR94] Izrail Solomonovich and Iosif Moiseevich Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, New York, NY, 1994.

[SSCJ98] Onn Shehory, Katia Sycara, Prasad Chalasani, and Somesh Jha. Agent cloning: An approach to agent mobility and resource allocation. *IEEE Communications*, 36(7):58–67, July 1998.

[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.

[Sta97] Dale O. Stahl. The inefficiency of auctions in dynamic stochastic environments. *University of Texas at Austin Department of Economics Working Paper Series*, June 1997. Working paper 9707.

[Sut68] Ivan E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, June 1968.

[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.

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

[WFL95] Michael P. Wellman, Matthew Ford, and Kenneth Larson. Path planning under time-dependent uncertainty. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 532–539, August 1995.

[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.

[Whi96] James E. White. Telescript technology: Mobile agents. General Magic White Paper, 1996.

[WS98] Anna Watson and M. Angela Sasse. Measuring perceived quality of speech and video in multimedia conferencing applications. In *Proceedings of the 6th ACM International Multimedia Conference*, Bristol, UK, September 1998.

[WWW98] Peter R. Wurman, Michael P. Wellman, and William E Walsh. The Michigan Internet AuctionBot: a configurable auction server for human and software agents. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 301–308, May 1998.

[WWW01] Peter R. Wurman, Michael P. Wellman, and William E Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 2001. To appear.

[YDFH98] Y. Yemini, A. Dailianas, D. Florissi, and G. Huberman. MarketNet: Market-based protection of information systems. In *Proceedings of the First International Conference on Information and Computation Economies*, pages 181–190, Charleston, SC, October 1998. ACM Press.

Index