The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms

Donald Francis Ferguson

CUCS-476-89

The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms

Donald Francis Ferguson

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY
1989

ABSTRACT

The Application of Microeconomics to the Design
of Resource Allocation and Control Algorithms

Donald Francis Ferguson

In this thesis, we present a new methodology for resource sharing algorithms in distributed systems. We propose that a distributed computing system should be composed of a decentralized community of *microeconomic* agents. We show that this approach decreases complexity and can substantially improve performance. We compare the performance, generality and complexity of our algorithms with non-economic algorithms. To validate the usefulness of our approach, we present economies that solve three distinct resource management problems encountered in large, distributed systems.

The first economy performs CPU load balancing and demonstrates how our approach limits complexity and effectively allocates resources when compared to non-economic algorithms. We show that the economy achieves better performance than a representative non-economic algorithm. The load balancing economy spans a broad spectrum of possible load balancing strategies, making it possible to adapt the load balancing strategy to the relative power of CPU vs. communication.

The second economy implements flow control in virtual circuit based computer networks. This economy implements a general model of VC throughput and delay goals that more accurately describes the goals of a diverse set of users. We propose *Pareto-optimality* as a definition of optimality and fairness for the flow control problem and prove that the resource allocations computed by the economy are Pareto-optimal. Finally, we present a set of distributed algorithms that rapidly compute a Pareto-optimal allocation of resources.

The final economy manages replicated, distributed data in a distributed computer system. This economy substantially decreases mean transaction response time by adapting to to the transactions' reference patterns. The economy reacts to localities in the data access pattern by dynamically assigning copies of data objects to nodes in the system. The number of copies of each object is adjusted based on the write frequency versus the read frequency for the object. Unlike previous work, the data management economy's algorithms are completely decentralized and have low computational overhead. Finally, this economy demonstrates how an economy can allocate *logical* resources in addition to physical resources.

CONTENTS

List of Illustrations

Acknowledgements

I wish to thank my advisor Yechiam Yemini for his guidance, support and friendship. I consider myself very lucky to have had the honor and pleasure of working with him. He has had a profound influence on this dissertation and on me. If I ever advise someone through a difficult challenge, I will try to be like him. If I am half as good as he, I will consider myself very successful. I am looking forward to working with him in the future.

I wish to thank my thesis committe: Zvi Galil, Michael Pinedo, Calton Pu, Mischa Schwartz and David Simcha-levi. I think that this dissertation has benefited from their comments and my discussions with them. I also want to thank my committee for their tolerance of the many phone calls and mail messages required to schedule the defense. I would especially like to thank Calton Pu for the many helpful conversations with him. Seldom does one meet a person whose advice is so consistently excellent.

I wish to thank the faculty, students and staff of the Computer Science Department. Despite some trying crises, I enjoyed working with and learning from them. Special thanks go to my friends and fellow suffering students Dannie Durand, Yoram Eisenstadter, David Glaser, Avraham Leff, Cecile Paris, Nat Polish, Jonathan Smith, Micheal van Biema and Moti Yung. I hope my friendship has helped them as much as their friendship helped me. I am grateful to Jed Schwartz and Alex Dupuy for their support of the DCC projects. Many of the results in this dissertation were greatly aided by Alex's patience and frequent explanations of Unix and NEST. From the faculty, I give special thanks to John Kender and Jonathan Gross. Finally, I must thank Germaine Leveque for guiding me through the maze of forms and instructions needed to accomplish anything at Columbia. I do not know how the department ever managed to function without her.

I have had the priveledge of working at IBM Research for most of my years in graduate school. I have learned a great deal from this experience. I wish to thank my managers Erich Port, John Pershing, Nagui Halim and Phil Rosenfeld for their support for my studies. I will always be thankful for my friends' influences on my work and on me. Special thank go to Michelle Kim (who introduced me to IBM; little did we know), Leo Georgiadis (for his guidance through the world on mathematics), and David Bacon (for letting me use his Suns). Finally, I must thank Christos Nikolaou who has been both an excellent advisor and a friend. He has contributed as much to this dissertation and to me as anyone.

I want to thank Michelle Sacks for her friendship and help. I am not sure I would have made it without her.

My in-laws Marilyn and Thomas Dente have been extremely helpful over the years. They were always there to supply encouragement, and their home was often a safe haven from the rigors of school work.

Finally, and most importantly, I thank my wife Kim. Her confidence in me and her support kept me going the countless times I wanted to stop. My finishing the program is as much her accomplishment as mine. I have not been able to be with her nearly as much as I wished and I am grateful that I can spend the rest of my life making this up to her.

To Kimberly

with love

1.0 Introduction

This dissertation focuses on the problem of allocating and sharing resources in a distributed system. We propose a novel approach to the design of resource allocation and control algorithms in distributed systems. Our thesis is that a distributed system should be structured as an economy of competitive *microeconomic* agents. Each agent independently tries to meet its individual goals, and no attempt is made to achieve system wide performance objectives. The competition between the agents computes a *competitive equilibrium.* In equilibrium, the global demand for each resource equals the supply of the resource, and the individual agents' demands can be satisfied. We show how a competitive equilibrium is an optimal, fair allocation of resources.

As distributed systems increase in size and power, the complexity of making resource management decisions grows dramatically. An important contribution of microeconomics to resource management in distributed systems is a methodology for limiting the the complexity of solving resource allocation problems. Our implementations of the three economies demonstrate how the new methodology limits complexity in two ways: 1) The system is inherently modular, and 2) All decision making is inherently decentralized.

To evaluate our methodology, we present three economies that solve classical resource management problems in distributed computer systems. These are 1) *load balancing* [29], 2) *flow control* in virtual circuit networks [9], and 3) *file allocation* [33]. We compare both the structure of the economies and their performance to traditional algorithms for solving these resource management problems.

The next section of this chapter elaborates on the complexity problems of designing resource management algorithms in distributed computer systems. This sets the motivation for pursuing the research presented in this dissertation. After the problems are discussed, a subsection gives an overview of the contributions that microeconomics can make to the study of distributed systems. Finally, the last two sections of this chapter present the main goals of this thesis, and an overview of its research contributions.

1.1 The Problem

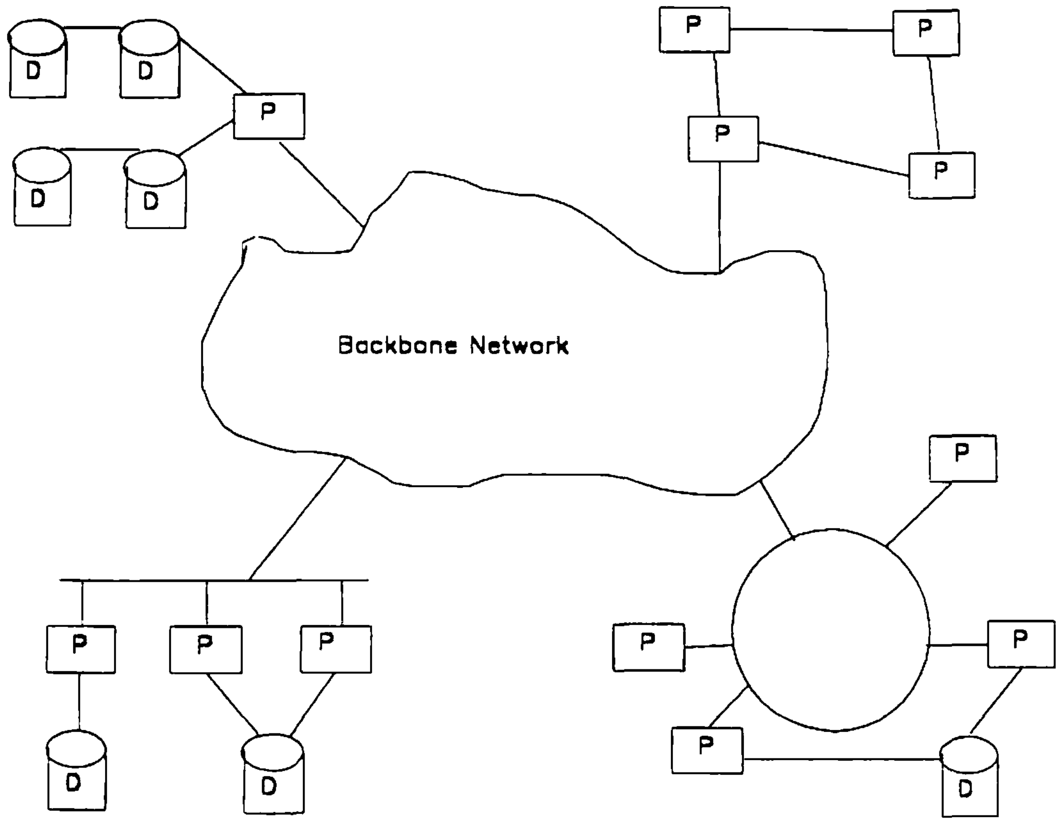


Figure 1. A Representative Distributed System: This figure shows a simple distributed system. It is composed of a long haul backbone network connecting multiple local area networks (point-to-point, bus, ring, etc). Each LAN contains processors (P), disks (D) and potentially many other physical and logical resources.

Figure 1 on page 2 shows a representative, large scale distributed system. This system is composed of a backbone network connecting several local area networks. Each of the local area network has its own interconnect architecture that connects processors and other hardware resources. The class of software applications supported by such a network is potentially infinite.

To fully achieve the potential of this distributed system, resources must be shared among the users of the system. Some resources, such as the backbone network, may be owned in common and must be shared for this reason. Furthermore, some users may be willing to share their resources with others in the system, provided that they are allowed to share resources owned by others in return. So, a set of algorithms is needed to control the allocation and sharing of resources within the system.

Computer science has a long history of pursuing research on resource sharing algorithms. In single processor systems, multiprocessing and virtual memory are two examples [23]. For computer networks, multiplexing on physical links, flow control and routing are areas that have been extensively studied [9, 81]. Most previous research on the design of resource sharing algorithms is characterized by at least one of two common features. The first feature is *centralization.* The information needed to make a resource sharing decision is gathered in a central location, the decision is made at this location and the result is distributed. For example, centralized algorithms for concurrency control [8] and CPU load balancing [29] have been studied. The second feature is *consensus.* The agents in the distributed system exchange information and attempt to achieve a common goal for the system as a whole. Consensus algorithms for routing and flow control [9], concurrency control [8] and load balancing [29] have also been proposed. For example, the common goal of a load balancing algorithm (centralized or consensus) could be either minimizing the average response or maximizing the total throughput.

Centralized and consensus based resource sharing algorithms attempt to compute an allocation of resources that optimizes some system wide performance metric. This is done in the hope that improving the global performance of the system will increase the "happiness" of the individual users. It may be the case that the global performance objectives do not accurately represent the goals of the diverse individual users. Defining global objectives that accurately reflect individual goals is becoming increasingly more difficult as the users of the system become more diverse. We show how a competitive equilibrium is an optimal allocation with respect to the diverse, conflicting individual goals.

As distributed systems increase in complexity, it is increasingly more difficult to design centralized or distributed consensus algorithms that effectively allocate the resources in the system. The complexity is caused by many problems. The first problem is one of *scale.* As the number of nodes, resources and applications composing the system increases, centralizing information becomes impractical. The communication overhead incurred by centralizing the information needed to make resource allocation decisions becomes prohibitively expensive. Furthermore, centralized decision making does not utilize the increased computational power of a distributed, multiple processor system.

Distributed, cooperative algorithms do not centralize information and do utilize the multiple resources. However, as the size of systems increase, the rate of change of important system state information needed for resource management also increases. It may be the case that the dynamic system state changes more rapidly than a consensus can be reached. Furthermore, the message cost of reaching a consensus increases with the size of the system.

A second cause of complexity is *heterogeneous users.* Increased CPU power, link speeds and memory capacity make it possible to support progressively more diverse applications in the system. Diversity in the community of users makes it difficult to define a global performance metric that accurately reflects the individual user goals. For example, in a database management system, the system may try to increase throughput by increasing concurrency. The network's goal may be low response time of the messages it processes. The underlying operating system may be attempting to maximize processor and memory utilization. It is not obvious that a single system goal can adequately cover these three sub-goals. Furthermore, there may be *conflicting goals.* For example, consider a distributed database management system supporting both simple update transactions and complex queries [21]. The submitters of the updates might be primarily concerned with response time, while those submitting complex queries may be concerned with throughput. It is difficult to define a system wide goal satisfying both types of users because it is typically the case that increased throughput causes increased response time. The same problem occurs in the network connecting the remote users to the DBMS system. Complex queries may require shipping large amounts of data through the network. So, throughput is important. The response time goals of the updates imply that network delays should be minimized, which conflicts with increasing throughput. A final problem with attempting to define a system wide goal is that the goal must be able to handle unforeseen new applications.

Finally, both centralization and cooperation are inherently *unstable.* In a system using a single processor to perform resource management decisions, a single failure can cripple the system. Cooperative algorithms face the same problem. If even one processor fails to cooperate, or does not cooperate as expected, the entire process can fail. Techniques such as Byzantine agreement [61] can be used to address this problem, but at the price of increased complexity and resource requirements.

This thesis proposes using concepts drawn from microeconomics to design resource sharing algorithms. In the next subsection, the new tools made available by economics are overviewed.

1.2 The Solution

Microeconomics can make two contributions to the study of resource sharing algorithms. The first is a set of tools for limiting complexity and improving performance. The second is a set of mathematical models that can yield new insights into resource sharing problems.

Microeconomics provides three tools that can be used to limit the complexity of making resource sharing decisions in a distributed system. The first is *decentralization.* The system is structured as a set of autonomous agents, and each agent has its own goals, plans for attaining these goals and endowment of resources that it can use. All decisions are made independently, and there is no attempt to collaborate on improving the system as a whole. The second tool offered by microeconomics is *competition.* Each agent selfishly attempts to maximize its own happiness. The agents do not cooperate and do not attempt to reach a consensus. The third tool is the use of *money*, and a *price system.* The price charged for a resource provides a single measure of the value of a resource compared to all others. Money can be used to define the relative importance, or priority, of the agents in the system. This helps deal with heterogeneity and conflicting goals.

The problems caused by scale are solved by decentralizing the decision making. When designing the resource management algorithms, it is not necessary to take the entire system state into consideration, and it is not necessary to obtain a coherent view of the system state. Instead, the algorithms' designer simply focuses on each agent as an individual. The goal is to design algorithms that maximize each agents satisfaction independently of all others. The overall efficient allocation of resources is the indirect result of the competition among agents. It is not intuitively obvious that local, selfish optimization yields a globally effective allocation of resources. One of the main results of this thesis is demonstrating that this assertion is true for problems in distributed computer systems.

Designing the system as a set of competing agents improves the system's software structure because the resulting algorithms are inherently modular. Agents may change their algorithms, and enter and exit the system without necessitating changes in any other agents. This will be demonstrated with examples in this dissertation.

Decentralization and competition also solve the problems caused by heterogeneity and diversity. It is no longer necessary to define a common system goal that adequately reflects the wants and desires of the diverse community. It is simply necessary to understand the goals of the individuals, and the economic competition computes a system state that is "optimal" with respect to the community of users. One of the major contributions of microeconomics is a new definition of optimality for resource allocation in a heterogeneous distributed system: Pareto-optimality [40]. The effectiveness of this definition is demonstrated in this dissertation.

Conflicting goals are resolved by competition among the agents whose goals conflict. The economy computes a resultant state that is an equilibrium point between the conflicting goals. The resulting state may not be optimal with respect to any individual, but is optimal with respect to the system as a whole. The economic competition in the system determines the prices charged for resources and services. These prices reflect the relative value of resources and provide a single measure of value for all resources. Agents are allocated money by some policy that is external to the economy, and this money is used to purchase the resources the agent demands. The initial endowment of money allocated to an agent defines its priority relative to other agents. The agents simply purchase the resources they desire at the given set of prices and use these resources as they see fit. There is no attempt to merge disparate goals into a single system wide goal.

Finally, competition is inherently stable. The failure of agent A in an economy can only harm A itself. Such a failure cannot cripple the system as a whole. The reliability of distributed systems structured using microeconomics concepts is a major benefit, but it is not addressed in this dissertation. This dissertation focuses on the improvements in performance and system structure that are achieved when these tools are used. Reliability of economies is a promising area of future work.

The mathematical models of microeconomics can provide new insights when computer systems are structured as competitive economies. These models can be used to prove optimality of resource allocations computed. It is also possible to prove the existence of an optimal equilibrium point. In this dissertation, we demonstrate the effectiveness of these models by applying them to the flow control problem in computer networks [9]. We also show that these models must be altered to accurately describe the problem being studied. Additionally, we cannot make the same assumptions as economists and it must be proven that the computer system possesses the properties typically assumed by the economic models. This is problem is explained in chapters 2 and 4.

1.3 Goals of the Dissertation

This dissertation has the following goals:

1. To explore the similarities between complex distributed computer systems and economies:
    a. To see which tools provided by economics can be applied to problems encountered in large distributed computer systems.
2. To transform these tools into effective resource sharing algorithms and apply them to problems in distributed computer systems.
    a. To compare the performance of these algorithms with non-economic algorithms.
    b. To determine the effects on complexity.
    c. To demonstrate the broad applicability of the tools by solving diverse resource sharing problems in distributed systems.
3. To study the applicability of mathematical economic models to distributed systems.
    a. To determine the assumptions that must be changed.

1.4 Dissertation Overview

Chapter two presents the economic background for the results presented in this dissertation. This background falls into two categories. First, we give a concise survey of the economic concepts related to the work of this dissertation, and we provide examples of the ways these concepts describe phenomena in distributed computer systems. These examples provide further motivation for our research. Secondly, we survey previous work on applying economic concepts to resource management problems in distributed computer systems. We compare and contrast this previous work with the results presented in this dissertation.

Chapter three presents the *Load Balancing Economy*. This economy allocates CPU resources to jobs submitted for execution in a distributed system. Since the economy performs load balancing across multiple processors, jobs migrate from processor to processor seeking CPU service. So, this economy also controls the allocation of communication resources to the submitted jobs. The load balancing economy demonstrates the applicability of decentralized decision making, competition and a price system to a traditional problem in computer science. In this economy, the processors selfishly attempt to maximize revenue and do not cooperate with other processors in an attempt to minimize average response time or maximize average throughput. Similarly, the jobs in the system compete with each other to obtain the CPU and communication resources they need to complete. This economy demonstrates how structuring distributed systems as an economy of microeconomic agents yields resource allocation algorithms that are inherently decentralized and modular. We will demonstrate that both the jobs and processors can change their goals and strategies transparently to all other agents.

The load balancing economy is extremely versatile and can easily be tuned to implement almost any load balancing strategy. The economy spans a broad spectrum of possible load balancing strategies. To evaluate the effectiveness of this economy, we compare its performance against a representative non-economic load balancing algorithm. This comparison shows why the load balancing economy is a significant contribution to the load balancing problem independently of exploring the applicability of economics to distributed systems. Finally, we discuss the effectiveness of wealth as a priority scheme, and the role of learning in determining the response time of individual jobs and the average response time of the system as a whole.

Chapter four presents the *Flow Control Economy*. This economy allocates communication bandwidth to virtual circuits in a computer network. The flow control economy illustrates the broad applicability of microeconomics to distributed systems by solving a vastly different problem from load balancing. The main contribution of this chapter is demonstrating the power of mathematical economics for the description of distributed system problems and to the design of effective resource management algorithms. Mathematical economics provides tools for proving that the resource allocations computed by an economy are Pareto-optimal. We prove that the algorithms of the flow control economy compute Pareto optimal allocations of communication resources. Pareto-optimality is a powerful definition for both optimality and *fairness* in heterogeneous distributed systems. We present a formalization of fairness metrics for the flow control problem and compare

Pareto-optimality with previous definitions of fairness. Finally, we prove the existence of a Pareto-optimal equilibrium for arbitrary networks.

Compared to previous work on flow control, the flow control economy implements a VC model that captures the diversity of users of virtual circuits in computer networks. Under this model, each virtual circuit's user is able to choose an independent throughput-delay goal. We also present a set of completely decentralized algorithm for computing optimal allocations of resources. These algorithms are iterative and rapidly converge to optimal allocations.

Chapter five presents the *Data Management Economy*. This economy implements parts of the Transaction Manager and Data Manager layers of a distributed data base system [8]. This chapter demonstrates the self-tuning behavior of economies. The data management economy improves mean transaction response time by adapting to the read/write ratio of transactions, and to localities in transaction reference patterns. This adaptation is effective over a broad range of parameters. We illustrate how this behavior occurs through examples. This economy also demonstrates that our microeconomic approach can control the sharing of logical resources (access to data) as well as physical resources. Finally, we point out some limitations of the current model.

Finally, chapter six is the conclusion. This chapter encapsulates the major results of the dissertation and also discusses future avenues of research opened by the results of this thesis. These fall into two categories. The first are direct extensions of the economies presented in this dissertation. The second avenue branches into new areas of research. The most promising of these is generalizing mathematical economics so that it can be applied to a broader set of distributed computer system models.

2.0 Economic Concepts and Related Work

This chapter serves two purposes. First, it contains a presentation of the economic concepts that are used in the three economies that are the research results of this dissertation. The discussion of economic concepts in this section is abstract. This is done so that the reader can use this section as a starting point to applying these concepts to other distributed resource allocation problems. The second purpose of this chapter is to provide a survey of previous work on applying economic concepts to problems in computer science. We feel that the material in this dissertation opens new avenues of research and compliments previous work.

The emphasis in this dissertation is on algorithms for allocating and controlling shared resources that improve system performance. This is a very broad area of research, but it is not the only area in which economic concepts can be used by computer science. Miller and Drexler [72] discuss other possible contributions that the field of economics can make to computer science as a whole. This work provides a very compelling motivation for the research in this area.

2.1 Economic Concepts and Terminology

This section provides an overview of the economic concepts and terminology used in the dissertation. For more detailed presentations of this material, Hildenbrand [40], Arrow and Hahn [5] and *The Handbook of Mathematical Economics* [1] are recommended. The material in this section is drawn from these sources.

2.1.1 Resources and Agents

The major similarity between computer systems and economies is the existence of a set of resources to be shared between multiple users. In economies, labor and coal are examples of physical resources. The physical resources in a computer system could be CPU time, storage and communication bandwidth. There may are also be *logical* resources in both economies and computer systems. In an economy, a patent is an example of a logical resource. In a computer system, a database lock can be considered a logical resource.

Formally, the set of resources in an economy is denoted $R_1, R_2, \ldots, R_N$. An *allocation* is a collection or bundle of the various resources, and is a vector in $\mathbb{R}^N$. If $\vec{x} = <x_1, x_2, \ldots, x_N>$ is an allocation, then the allocation contains x_i units of resource R_i. Figure 2 on page 12 depicts a simple computer system with two resources. R_c represents the CPU and R_m represents main memory. An allocation $\vec{x} = <10ms, 128\ Kbytes>$ contains 10 milliseconds of CPU time and 128 Kbytes of main memory.

In addition to resources, an economy contains a set of agents $A_1, A_2, \ldots, A_M$, which are the active participants in the economy. There are two types of agents in an economy. The first type is a *supplier* (or *producer*). If agent A_i is a supplier, it controls some resources that it makes available to be sold to the second type of agents, which are *consumers*. In a computer system, the operating system (or processor) can be modeled as a supplier. It sells CPU time and memory to users of the system, which are the consumers. Application programs and transactions are examples of consumers in a computer system. The sets of suppliers and consumers are not necessarily disjoint. For example, a database management system consumes resources (CPU, Memory) needed to perform its functions. It also provides access to data and locks needed by user submitted transactions.

2.1.2 Prices, Budgets and Demand

A *price system* (or *price vector*) is a vector $\vec{p}$ in $\mathbb{R}^N$ with $p_i \geq 0$. p_i is the price per unit for resource R_i. In economic theory, $\vec{p}$ is known by all agents. In a computer system, communication delays may make total knowledge of $\vec{p}$ impossible. The agents must be robust in the face of limited knowledge. This problem is addressed in different ways in the economies of chapters 3, 4 and 5.

Given a price system $\vec{p}$ it is possible to determine the cost or value of an allocation of resources $\vec{x}$. This is simply,

$$\vec{p} \cdot \vec{x} = \sum_{i=1}^{N} (p_i \cdot x_i).$$

For example, in Figure 2 on page 12, if CPU time is \$1 per millisecond and memory is \$0.01 per byte, then the allocation $\vec{x} = <10ms, 1\ Kbyte>$ has value (costs) \$20.24. Since consumers must purchase resources, they must be allocated some money when they enter the system. The initial wealth, or *endowment* of agent A_i is denoted W_i.

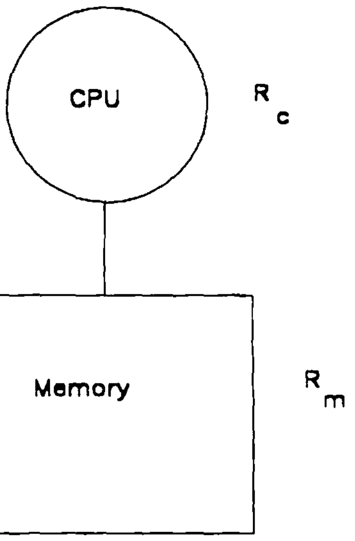


Figure 2. Resources: A simple economy with two resources. These resources are the CPU and the main memory.

Given a price system and an endowment, the set of allocations agent A_i can afford is called its *budget set* and is denoted

$$B(A_i, \vec{p}) = \{\vec{x} \mid \vec{p} \cdot \vec{x} \le W_i \text{ and } x_k \ge 0\}.$$

Figure 3 on page 13 shows a simple computer systems. This system can be modeled as an economy of four agents and two resources. The resources are CPU time and main memory, and these are supplied by the operating system agent (A_0). There are three application programs in the system. They are the following:

1. A database management system (DBMS) : This program has an endowment of $1,499,000.
2. A numerical algorithm (NA) : This program has endowment of $401,000.

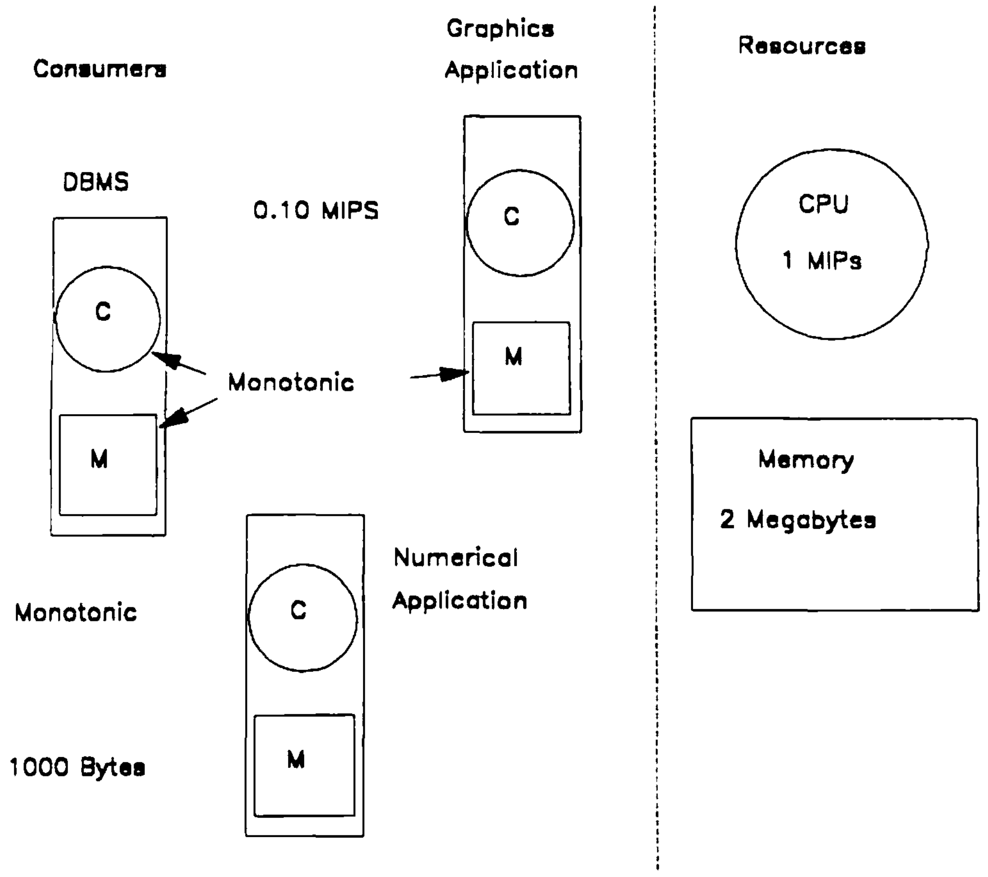


Figure 3. A Simple Computer System: This system contains two resources (CPU and Memory) and four agents, 1 supplier and 3 consumers of CPU and Memory. In this figure, the values next to the agents represent their demand for resources. For example, the graphics application demands 100,000 instructions per second and as much memory as possible. Monotonic means that more of a resource is always better.

3. A graphics application (GA) : endowment of \$1,100,000.

If the current price vector is

$$\vec{p} = < \frac{\$1}{\text{IPS}}, \frac{\$1}{\text{byte}} >,$$

where IPS is instructions per second, then the allocation

$$\vec{x} = < 1 \text{ MIPS}, 10^5 \text{ bytes} >$$

is in $B(GA, \vec{p})$, but

$$\vec{x} = < 1 \text{ MIPS}, 10^5 \text{ bytes} + 1 >$$

is not.

The endowment of an agent and the current price vector define the agent's budget set which is the set of resource allocations it can afford. Obviously, not all elements are equally "desirable," however. In the example of Figure 3 on page 13, we could we could assume most agents would rate allocation

$$\vec{x} = < 1 \text{ MIPS} , 10^6 \text{ bytes} >$$

less desirable than allocation

$$\vec{y} = < 1.5 \text{ MIPS} , 2 \times 10^6 \text{ bytes} > .$$

In an economy, each agent has a *preference relation* that formally defines its individual notion of desirability. Formally, agent A_i's preference relation is denoted $\succcurlyeq_i$ and is a binary relation on the set of possible allocations. If for two allocations, $\vec{x} \succcurlyeq_i \vec{y}$, then A_i rates allocation $\vec{x}$ at least as desirable as $\vec{y}$. If it is not also the case that $\vec{y} \succcurlyeq_i \vec{x}$, then $\vec{x}$ is *strictly preferred* to allocation $\vec{y}$. Strict preference is denoted $\vec{x} \succ_i \vec{y}$. If we have $\vec{x} \succcurlyeq_i \vec{y}$ and $\vec{y} \succcurlyeq_i \vec{x}$, then A_i is *indifferent* between $\vec{x}$ and $\vec{y}$. Indifference is denoted $\vec{x} \sim_i \vec{y}$.

In the example of Figure 3 on page 13, assume that the numerical application requires only 10^3 bytes of storage to execute and needs as much CPU as possible, i.e. - the application is extremely CPU bound. Assume that if more then 10^3 bytes is purchased, it will go unused. Then, for the following three allocations:

1. $\vec{x} = < 1 \text{ MIPS} , 10^3 \text{ bytes} >$
2. $\vec{y} = < 1.5 \text{ MIPS} , 10^3 \text{ bytes} >$
3. $\vec{z} = < 1 \text{ MIPS} , 10^4 \text{ bytes} >$

we would have

1. $\vec{y} \succ_{NA} \vec{x}$

2. $\vec{y} \succ_{NA} \vec{z}$

3. $\vec{z} \sim_{NA} \vec{x}$

Given an agent A_i and a price system $\vec{p}$, the budget set may have many elements. The agent chooses an optimal element in $B(A_i, \vec{p})$, where optimality is defined by $\succcurlyeq_i$. The set of optimal elements in $B(A_i, \vec{p})$ is called the *demand set* and is denoted $\Phi(A_i, \vec{p})$. Formally,

$$\Phi(A_i, \vec{p}) = \{\vec{x} \mid \vec{x} \in B(A_i, \vec{p}) \text{ and } \vec{x} \succcurlyeq_i \vec{y}, \forall \vec{y} \in B(A_i, \vec{p})\}.$$

In the example of Figure 3 on page 13, assume that $\vec{p} = <1, 1>$. The numerical application's demand set has a single element

$$\Phi(A_{NA}, \vec{p}) = \{ < 4.0 \times 10^5 \text{ IPS }, 10^3 \text{ bytes } > \}.$$

The preference relation and the endowment are key tools used to handle the diversity present in a large, distributed system. Each agent is free to rank allocations of resources as it sees fit, and there is no need to define a global optimal allocation. To demonstrate the modeling of diversity, we expand on the example of Figure 3 on page 13. This example will also be used in the following sections. In the example of Figure 3 on page 13, assume that the DBMS application is interested in maximizing throughput. Furthermore, assume that the DBMS executes 10,000 instructions and makes 5 data references per transaction. The size of the database is 10^8 bytes. If a data object referenced is in the cache, then it takes an insignificant amount of time to perform the access. Otherwise, 100 milliseconds are needed to fetch the object from secondary storage. Furthermore, assume that the DBMS executes the transactions serially, and that it is suspended while an I/O is being performed. Given these assumptions and assuming that the accesses are uniformly distributed, if $\vec{x} = <x_1, x_2>$ is an allocation of resources to the DBMS, then the expected delay per transaction is

$$D(\vec{x}) = \frac{10^4}{x_1} + (5 \cdot 10^{-1})[\frac{(10^8 - x_2)}{(10^8)}]$$

and the throughput is $T(\vec{x}) = \frac{1}{D(\vec{x})}$. In this case, for two allocations $\vec{x}$ and $\vec{y}$, we have

$$\vec{x} \succcurlyeq_{DBMS} \vec{y},$$

if, and only if,

$$T(\vec{x}) \geq T(\vec{y}).$$

The third application in this example is a graphics application. Assume that this application is memory bound. It needs only CPU resources of 10^5 instructions per second, but wants as much memory as possible. Assume that if this agent does not get 10^5 IPS, it cannot execute at all. Let $\vec{x}$ and $\vec{y}$ be two allocations. This agent's preference relation is given by the following rules:

1. If $x_1 < 10^5$ and $y_1 < 10^5$, then $\vec{x} \sim_{GA} \vec{y}$. In both cases, the application cannot execute at all.
2. If $\vec{x} \geq 10^5$ and $\vec{y} \geq 10^5$, then $\vec{x} \succcurlyeq_{GA} \vec{y}$, iff $x_2 \geq y_2$.

This example highlights two aspects of preference relations. The first is that they are very general and can adequately model a very diverse set of resource needs in a computer system. Furthermore, an agent's preference relation can be defined independently of those of other agents. The second aspect is that to adequately define an agent's preference, it is necessary to understand the individual agent's resource usage in detail. The first aspect is a benefit, but the second can be troublesome. For the remainder of this thesis, we assume that the resource demands of the agents are completely known by the agents. Extending this work to other scenarios is a topic for future research.

2.1.3 Pareto-Optimality and Fairness

The goal of resource management in both economies and computer systems is to compute an "optimal" allocation of resources to agents. If each agent chooses its own definition of optimality, it is not clear how to define an optimal allocation for the system as a whole. One approach to dealing with this problem is to define a *utility function* for each agent, and then maximize some function of the individual utilities. For an agent A_i , a utility function $u_i(\vec{x})$ is a function that maps allocations into $\mathbb{R}$ with the property that $\vec{x} \succcurlyeq_i \vec{y}$, if and only if, $u_i(\vec{x}) \geq u_i(\vec{y})$. In the example of Figure 3 on page 13, $T(\vec{x})$ is a utility function for the DBMS agent. For any preference relation, it is possible to define a corresponding utility function.

Given a utility function for each agent, it is possible to define a global objective function by combining the individual utilities. For example, we could maximize the sum or product of the individual utilities. There are several problems with this strategy. First, the individual utility functions could reflect drastically different goals. For example, $u_1(\vec{x})$ could be throughput for agent A_1, while $u_2(\vec{x})$ could be -1 times the average response time. Secondly, maximizing the sum or product of utilities can lead to cheating. An agent may incorrectly report its utility function to obtain more resources. Finally, maximizing global performance metrics can mean assigning 0 resources to some agent. This is demonstrated in chapter 4.

The definition of optimality used in economics is *Pareto-optimality*. Intuitively, a set of allocations $\vec{\phi}^1, \vec{\phi}^2, \ldots, \vec{\phi}^N$ of resources to agents $A_1, A_2, \ldots, A_N$ is Pareto-optimal if no agent A_i can be given a better allocation without forcing a worse allocation on another agent. Formally, a set of allocations $\vec{\phi}^1, \vec{\phi}^2, \ldots, \vec{\phi}^m$ to set of agents $C = \{A_1, A_2, \ldots, A_m\}$ is *feasible* if

$$\sum_{i \in C} (\vec{\phi}^i \cdot \vec{p}) \le \sum_{i \in C} W_i.$$

In other words, as a group, the agents can afford the allocation. The set of agents C can improve on the allocations $\vec{\phi}^1, \vec{\phi}^2, \ldots, \vec{\phi}^m$ if there exists another allocation $\vec{\beta}^1, \vec{\beta}^2, \ldots, \vec{\beta}^m$ meeting the following properties:

1. $\vec{\beta}^i \succcurlyeq_i \vec{\phi}^i$, for all $A_i \in C$. That is, everybody is at least as happy.
2. $\vec{\beta}^j \succ_j \vec{\phi}^j$, for at least one $A_j \in C$. In other words, some agent strictly prefers its new allocation.
3.

$$\sum_{i \in C} (\vec{\beta}^i \cdot \vec{p}) \le \sum_{i \in C} W_i.$$

The new allocation is affordable.

Given these concepts, it is possible to formally define Pareto-optimality.

> **Definition** : A set of allocations $\vec{\phi}^i$ to the N agents in an economy is **Pareto-optimal** if no subset of agents can improve on their allocation.

As an example of a Pareto-optimal allocation, return to the economy of Figure 3 on page 13. The following set of allocations is Pareto-optimal:

1. $\vec{\phi}^{DBMS} = < 5 \times 10^5, 10^6 - 10^3 >$.
2. $\vec{\phi}^{NA} = < 4 \times 10^5, 10^3 >$.
3. $\vec{\phi}^{GA} = < 1 \times 10^5, 10^6 >$.

In this set of allocations, no single agent can unilaterally surrender resources without receiving a strictly less desirable allocation. So, for a subset to improve on its allocation, trading must occur. However, agent A_{GA} cannot trade away CPU to get more memory and agent A_{NA} cannot trade away memory for CPU. So neither can be in a subset that improves on its allocation. This in turn implies that no subset can improve, and the allocation is Pareto-optimal.

As an example of a set of allocations that is not Pareto optimal, consider the following allocations:

1. $\vec{\phi}^{DBMS} = < 5 \times 10^5, 10^6 - 10^3 >$.
2. $\vec{\phi}^{NA} = < 1 \times 10^5, 5 \times 10^5 + 10^3 >$.
3. $\vec{\phi}^{GA} = < 4 \times 10^5, 5 \times 10^5 >$.

By cooperating, agents A_{NA} and A_{GA} can improve on their allocations. A_{GA} can trade 3×10^5 IPS to agent A_{NA} for 5×10^5 bytes of memory. This improves both agents' allocations. Agent A_{GA} gets the memory necessary to execute, so it is happier. Agent A_{NA} meets its memory requirement and receives more CPU, so it is happier.

The main advantage of Pareto optimality as a definition for optimal resource allocations is that it does not require any coordination between the preferences or utilities of the various agents. Each agent defines its preferences as it sees fit. This limits the complexity of the system by breaking the resource allocation problem into N independent subproblems.

There are disadvantages to the use of Pareto-optimality as defined in the examples above. First, in the second example, to improve on their allocations agents A_{NA} and A_{GA} had to barter and cooperate to exchange resources for improving their lots. Bartering and trading are complex operations. For example, a protocol for describing proposed trades is necessary. The second problem is that Pareto-optimal allocations can be extremely *unfair*. For example, the following set of allocations is Pareto-optimal:

1. $\vec{\phi}^{DBMS} = < 10^6, 2 \times 10^6 >$.
2. $\vec{\phi}^{NA} = < 0, 0 >$.
3. $\vec{\phi}^{GA} = < 0, 0 >$.

Agent A_{DBMS} cannot improve its allocation through trading, and agents A_{NA} and A_{GA} cannot have their allocations improved without decreasing the desirability of A_{DBMS}'s allocation.

The explicit use of money in the economy eliminates these two problems. First, an agent only needs to state its demands at the given prices. There is no need to reveal preferences and negotiate with other agents. Secondly, Pareto-optimality together with prices implements a very rigorous definition of *fairness*. For two agents A_i and A_j, the relative value of their demanded allocations at any price system $\vec{p}$ will be exactly $\frac{W_i}{W_j}$. The agents' endowments can be used to assign relative priority, and the price system reflects the relative value of resources. So, each agent gets exactly it fairs share. For example, let $p = < 1, 1 >$ be the current prices in the example of Figure 3 on page 13. Assume that the following allocations are demanded:

1. $\vec{\phi}^{DBMS} = < 5 \times 10^5, 10^6 - 10^3 >$.
2. $\vec{\phi}^{NA} = < 4 \times 10^5, 10^3 >$.
3. $\vec{\phi}^{GA} = < 1 \times 10^5, 10^6 >$.

We have

$$\frac{\vec{p} \cdot \vec{\phi}^{DBMS}}{\vec{p} \cdot \vec{\phi}^{GA}} = \frac{W_{DBMS}}{W_{GA}}.$$

So, the relative value of the allocations demanded is exactly the same as the relative value of the agents endowments, or priorities.

2.1.4 *Pricing Policies*

The previous subsections focused on the consumer agents. The main role of supplier agents is setting prices. Three policies for price setting are described in this section.

2.1.4.1 Tatonement

If the demand for resource R_i at prices $\vec{p}$, denoted $D_i(\vec{p})$, is greater than the supply S_i, the resource is undervalued and its price should be increased. If $D_i(\vec{p}) < S_i$, then R_i is too expensive and its price should fall. This policy for setting prices is a *tatonement* process, and it explicitly attempts to compute a *competitive equilibrium* in which supply equals demand for all resources. A competitive equilibrium has two desirable properties. The first is that the demand of the individual agents at the equilibrium prices $\vec{p}^*$ can be met, and all resources are fully utilized. The second property is that the allocation of resources in a competitive equilibrium is provably Pareto-optimal.

Define the *excess demand function* for resource R_i as $Z_i(\vec{p}) = D_i(\vec{p}) - S_i$. It is important to note that $D_i(\vec{p})$ is a function of the entire price system, and not just p_i. For example, consider the graphics application A_{GA} of Figure 3 on page 13. This application cannot execute without being allocated 10^5 instructions per second, and the application has W_{GA} = \$1,100,000. If the price for CPU is greater than \$11, agent A_{GA} cannot execute and will not demand any memory. If the price for CPU falls to \$10, A_{GA} will demand 10^5 IPS. This costs \$1,000,000 and A_{GA} will use the remaining \$100,00 to purchase memory. So, the demand for memory will increase even if there is no change in the price of memory.

Despite the fact that $D_i(\vec{p})$ is a function of $\vec{p}$ and not just p_i, in the tatonement process, the supplier of R_i only updates p_i based on $Z_i(\vec{p}) = D_i(\vec{p}) - S_i$. The seller of R_i may not have any control over the prices for the other resources since other agents may supply them.

The seller's algorithm for updating p_i based on $Z_i(\vec{p})$ is given by the formula:

$$p_i = p_i + (p_i \cdot k) \cdot \left(\frac{Z_i(\vec{p})}{S_i} \right)$$

where k is a constant. This algorithm makes the change in price proportional to the relative difference between supply and demand, and the current price.

2.1.4.2 Auctions

The tatonement process implicitly assumes that the demand for a resources is a smooth function of the price system $\vec{p}$. This may not always be the case. For example, consider the graphics application of Figure 3 on page 13. If the price for CPU is $11.01 per instructions per second, agent A_{GA} cannot afford to meet its CPU goal and will not demand any of this resource. If, however, the price falls to $11 per IPS, agent A_{GA} instantly demands 10^5 IPS. The total demand for a resource is the sum of the individual demands. So, in this example the demand for CPU is not a smooth function of the price system.

A second assumption for tatonement is that the resources are infinitely divisible. Memory is almost infinitely divisible because it is composed of a very large number of very small units, each of which can be sold separately. A lock on a record in a database is an example of a resource that is not infinitely divisible. The lock is either held, or not held. This is an example of a resource which is a discrete, indivisible quantity.

If the resource is not infinitely divisible and the demand does not vary smoothly with prices, an alternative to tatonement must be used. In this section, we discuss auction models for setting prices and selling resources.

Assume that resource R_i comes in a fixed, discrete supply S_i. This resource's producer P_j can hold an auction to sell R_i. P_i 's goal is to maximize revenue by selling the resource at the highest price a consumer agent is willing to pay. The strategy of the consumers is twofold. First, a consumer A_k that wants to purchase R_i attempts to obtain the resource at the lowest price. A_k also must consider the usefulness of resource R_i relative to other resources and their prices. This determines how much A_k is willing to pay for R_i.

There are many auction models for selling resources [24]. In this section, we briefly describe three models. The auction models used in the load balancing economy are based on the policies described here.

The first auction modeled is a *sealed bid auction*. When resource R_i becomes available for sale, the producer P_j announces the resource's availability and solicits bids. In parallel, each agent A_k determines how much it is willing to pay for this resource, and sends a bid containing this amount to

P_i . Agents are not aware of the amounts bid by other consumers. After all bids have been received, P_j opens the bids and awards R_i to the agent A_w that submitted the highest bid and collects the amount bid from A_w . Figure 4 on page 23 depicts a sealed bid auction (as well as two other models described below).

A second model is the *Dutch auction*. This model is controlled by the seller. P_j sets a high price p_i for the resource and determines if any consumer will pay this price. If not, P_j gradually lowers p_i until some consumer submits a bid. Figure 4 on page 23 also shows an example of the Dutch auction model.

The third auction model is an *English auction* (Figure 4 on page 23). Under this model, an agent A_k submits an initial bid b_k . The seller P_j sets p_i to b_k and waits for further bids. If no other consumer is willing to pay higher than b_k, then A_k wins the auction and is allocated the resource. If, however, another agent A_l submits a bid $b_l > b_k$, the price p_i is set to b_l and the process repeats. The auction terminates when no agent is willing to pay more than the current price.

The major advantage of the sealed bid auction is low overhead because only one round of bidding takes place. In both the Dutch and English auctions multiple rounds may be required to sell the resource. For every round, each agent must be allowed to apply its behavioral rules to determine if, and how much it will bid for the resource. Each of these rounds takes time and increases the overhead of the resource allocation process. The main advantage of the Dutch and English auctions is that the sale price of a resource more accurately reflects the number and the wealth of the agents competing for it.

2.1.4.3 Variable Supply Models

The tatonement and auction models assume that the supply of each resource R_i is fixed at S_i. This is not necessarily the case, and the data management economy in chapter 5 contains resources for which the supply can contract and expand dynamically. Since the supply can expand and contract to match the demand, the seller can attempt to set the price p_i to the value that maximizes its revenue. That is, the producer P_j attempt to find p_i that maximizes

$$p_i \cdot D_i(\vec{p}).$$

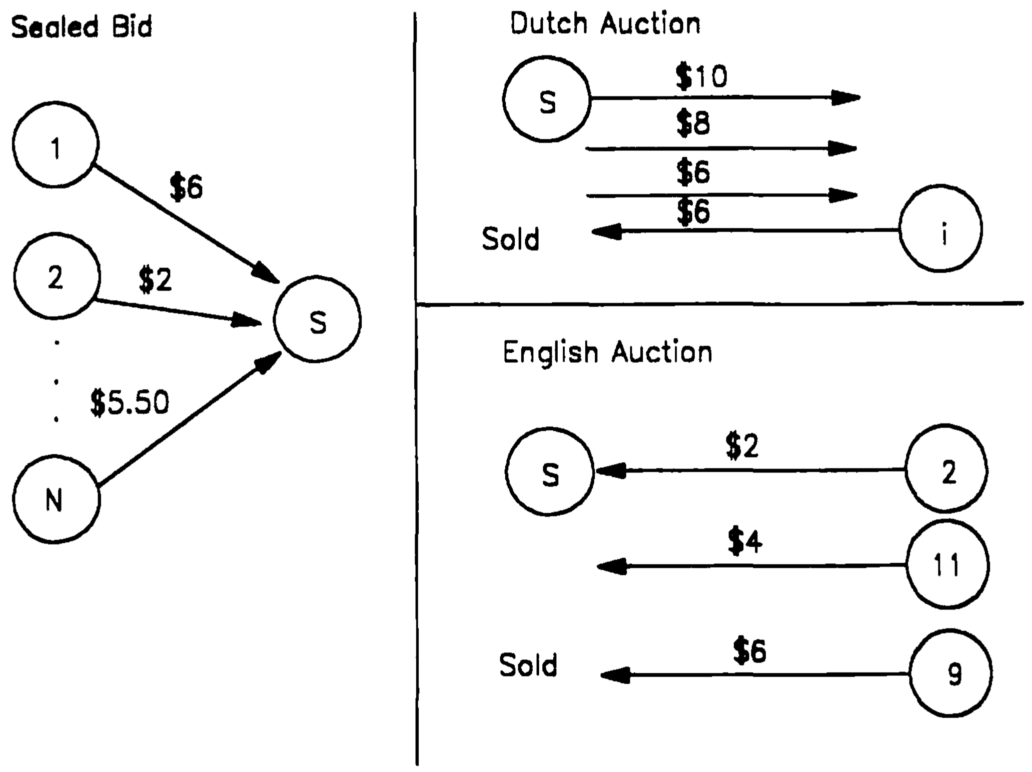


Figure 4. Auction Models: This figure represents three different auction models. In a *sealed bid auction*, all bids are submitted in parallel and the highest bidder wins. In a *Dutch auction*, the seller progressively lowers the price of the resource until a bid is submitted. Under the *English auction* model, the consumers individually submit progressively higher bids until the highest bidder is determined.

Determining p_i^* that maximizes revenue is an extremely difficult task. There are two reasons for this. First, the demand for R_i is a function of all resource prices, not just p_i. Secondly, the demand function $D_i(\vec{p})$ can be any arbitrary function of $\vec{p}$ and is not necessarily well behaved. Figure 5 on page 25 depicts three possible demand functions in a two consumer (A_1, A_2), one resource (R) economy. The three cases are the following:

1. A_1 and A_2 both have wealth W and demand as much of resource R as they can afford, up to limit 1.5. This is case A in the figure.
2. A_1 and A_2 both have wealth W and demand 1 units of the resource, if they can afford it. If not, their demand demand is 0. (Case B)

3. A_1 has wealth W_1 and A_2 has wealth W with $W_1 < W$. Each agent demands 1/2 unit if affordable. (Case C)

The unpredictability of the demand functions, and the fact that $D_i(\vec{p})$ is a function of $\vec{p}$ (as opposed to only p_i) make it practically impossible to find p_i^* maximizing revenue. Our experiments with the data management economy reinforce this intuition. To deal with this problem, a simple heuristic is used. Instead of making demand a function of price, the resource price is a function of demand. If the current demand for resource R_i is D_i, the seller sets p_i to $C(D_i)$, where C is the seller's price function, and each seller is free to individually choose C. Figure 6 on page 26 shows an example in which $C(D_i)$ is a simple function of D_i.

The main advantage of this heuristic is simplicity. The data management economy in chapter five will shows that even if simple functions are used, the economy can still exhibit extremely sophisticated and effective behavior. The main disadvantage of this heuristic is lack of theoretical foundation.

2.2 Related Work

This section presents a concise survey of previous work that has applied concepts from economics to resource control problems in computer systems. This previous work is compared and contrasted with the results presented in the dissertation.

Some previous work has applied economic concepts to the resource control problems studied in this dissertation. The *File Allocation Problem* [33, 88], which is closely related to the data management problem studied in Chapter 5, has been previously studied using economic concepts. Economic and game theoretic approaches have been applied to the flow control problem of chapter 4. Finally, economic algorithms for the task allocation problem, which is related to Chapter 3's load balancing problem, have been proposed. In these three cases, we additionally compare previous work with these economies purely as solutions to given problems.

In this dissertation, we are primarily interested in demonstrating that economic concepts can be used to design resource control algorithms that improve performance and decrease complexity. Using economic approaches creates a new set of problems that must be addressed, however. For example, the economies in this dissertation all use money as a medium for selling and purchasing

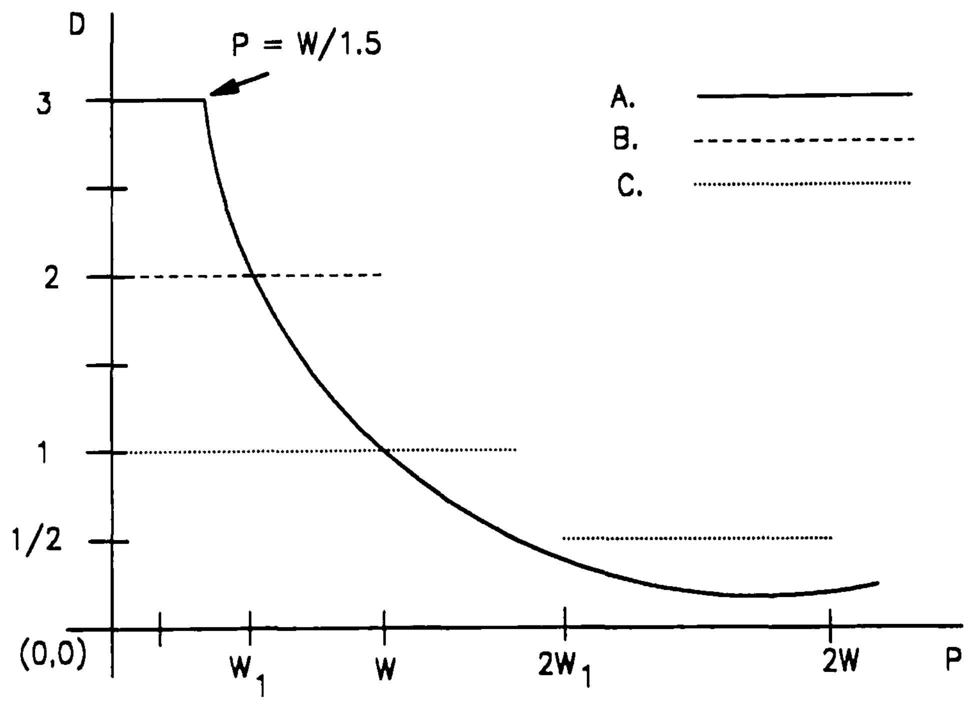


Figure 5. Demand Functions: This figure presents three possible demand function in a two consumer, 1 resource economy. The X-axis represents the resource's price and the Y-axis represents the total demand of both agents. The cases are: A) Each agent has wealth W and demands as much as is affordable up to 1.5 units. B) Each agent has wealth W and demands 1 unit if affordable. C) Agent A_1 has wealth W_1 and agent A_2 has wealth W. Each demands 1/2 if affordable.

resources. This raises the possibility of agents counterfeiting or embezzling money. When systems are structured using competition, honesty becomes a major issue. Drexler and Miller [24] examine issues of security and agent integrity in computer systems based on economic concepts. They present high level algorithms for allocating CPU and main memory that deal with potentially illegal behavior of agents. The economies presented in this dissertation will assume that all agents are honest. Dealing with agents that are less than scrupulous is an area for future research.

2.2.1 The File Allocation Problem

Kurose and Simha [58-60] have applied algorithms based on mathematical economics to the *File Allocation Problem*. In their model, there is a distributed system composed of N independent

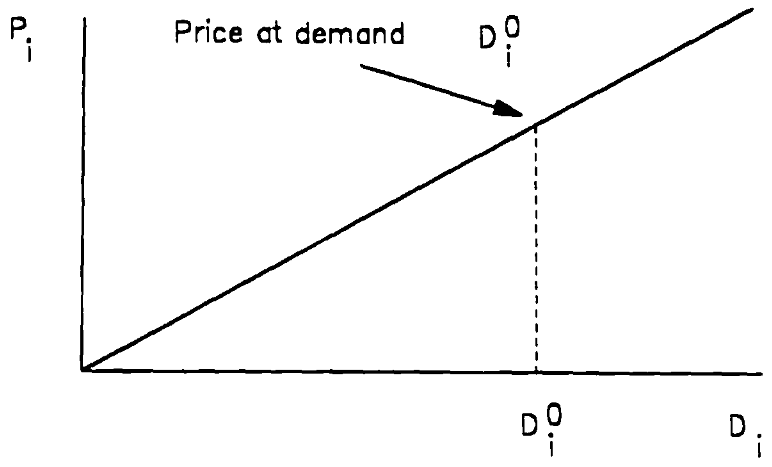


Figure 6. Demand Based Prices: This figure represents a simple heuristic for setting the price of a resource based on the current demand for it.

processors $P_1, P_2, \ldots, P_N$. These processors are connected by point-to-point links. For any processors P_i and P_j, there is a logical path composed of one or more links connecting the processors.

There is a single file resource X that must be assigned to processors in the system. Each processor P_i is to receive some fraction x_i of the file resource X. If X is a file system, then the fraction x_i could represent a subset of the files in the file system. If the resource X is a single file, x_i could represent a subset of the records in the file. The problem is to choose the fractions $x_1, x_2, \ldots x_N$ in a way that optimizes some performance measure. The x_i are percentages of the total resource assigned to the processors. So, we have

1. $\sum_{i=1}^{N} x_i = 1$.
2. $0 \leq x_i \leq 1$.

Accesses to the file resource are generated at all processors in the system. As a simplification, it is assumed that the accesses are uniformly distributed over the entire file resource. So, x_i is also the probability that a file access submitted anywhere in the system will be routed to P_i for processing.

It is assumed that there is an analytic model describing the performance of the underlying distributed system. This model is used to define a function that will be optimized by the choices of the x_i. The model is defined by the following parameters:

1. λ_i - The rate at which file accesses are generated at P_i. This is assumed to be a poisson process. The network wide access arrival rate is $\lambda = \sum_{i=1}^{N} \lambda_i$.
2. c_{ij} - The *communication cost* of transmitting an access from P_i to P_j and returning the response to P_i.
3. C_i - The average system wide *communication cost* of making an access at node P_i. This is the weighted average over all nodes and is

$$C_i = \sum_{j=1}^{N} \frac{\lambda_j}{\lambda} \bullet c_{ji}.$$

4. $\frac{1}{\mu}$ - The average processor service time for an access request, which is exponentially distributed.

Given these definitions, it is possible to derive the expected service time of an access operation at processor P_i. This is the expected service time plus the expected queueing delay and is given by

$$T_i = \frac{1}{\mu - \lambda x_i} .$$

λ is the system wide arrival rate, and x_i is the probability of an access being directed at the fraction of the file resource stored at P_i. So, the arrival rate at P_i is λx_i.

Let K be a constant that defines the relative cost of communication versus computation. The total cost of an assignment, including processor and communication costs is

$$\sum_{i=1}^{N}(C_i + \frac{K}{\mu - \lambda x_i})x_i.$$

This is taken to be the *utility* of an assignment $x_1, x_2, \ldots, x_N$. To keep the economic flavor, the algorithm attempts to maximize the function

$$U(x_1, x_2, \ldots, x_N) = (-1.0)\sum_{i=1}^{N}(C_i + \frac{K}{\mu - \lambda x_i})x_i.$$

The algorithm for maximizing U is motivated by algorithms from mathematical economics [38, 39, 41] that do not use a price system. Initially, each processor P_i is arbitrarily allocated some fraction x_i of the file resource. This initial allocation must be *feasible*, i.e - $0 \le x_i \le 1$ and $\sum_{i=1}^{N} x_i = 1$. The processors then cooperate by exchanging file resources in an attempt to improve the utility of the system has a whole. The algorithm is an iterative gradient algorithm. In the basic version of the algorithm each iteration is:

Step 1: Each node P_i computes

$$U'_i = \frac{\partial U(x_1, x_2, \ldots, x_N)}{\partial x_i}.$$

Step 2: Processor P_i sends U'_i to all other processors.

Step 3: The change in processor P_i's allocation of the resource X is given by

$$\Delta x_i = \alpha(U'_i - \frac{1}{N}\sum_{i=1}^{N} U'_j).$$

In this step, if U'_i is greater than the average marginal utility over the system as a whole, increasing P_i's share by some small amount and decreasing the fraction assigned to the remainder of the system by the same amount will increase the system wide utility. The actual amount P_i receives is proportional to difference between U'_i and the average marginal utility, and is regulated by a *step size* α .

Finally, the algorithm terminates if

Step 4:

$$|U'_i - U'_j| \leq \varepsilon$$

for all i and j. If this is not the case, the process is repeated starting at step 1.

The basic algorithm above must be modified in step 3 to ensure that processor P_i does not receive $x_i < 0$. This is done by excluding P_i from the update of step 3 if including it would result in a negative allocation.

Step 2 can be computed in parallel at all processors using only local information. This is due to the fact that

$$\frac{\partial U(x_1, x_2, \dots x_N)}{\partial x_i}$$

contains only x_i and constants. Step 3 requires complete information, however. Each processor P_i must know U'_j for all j to compute the average marginal utility. This limits the effectiveness of the algorithm as a decentralized algorithm for allocating resources. Finally, the termination detection of step 5 requires global information.

To avoid the overhead of requiring complete information in step 3, a distributed version of the algorithm is proposed. It is based on pair wise exchange of file resources in an attempt to improve the utility of the system as a whole. In the distributed version, processor P_i iteratively "pairs" with neighboring processors. When P_i pairs with neighbor P_j, they compute the δx_i and δx_j that maximizes δU subject to $\delta x_i - \delta x_j = 0$. Kurose and Simha present an analytic formula that defines the δx_i and δx_j. This formula contains only U'_i , U'_j, locally computed second partial derivatives and constants. This eliminates the need for global knowledge in step 3 of the basic algorithm. This modification does not solve the problem of termination detection, however. This still requires knowledge of U'_i for all processors P_i .

The main advantage of this algorithm is that it is possible to prove that it possesses the following properties:

1. *Optimality* : The algorithm computes feasible x_i that maximize the utility function $U(x_i, x_2, \dots, x_N)$.
2. *Convergence* : The algorithm converges *monotonically* to an optimal allocation. The proof sets an upper bound for the step size α. With this value, simulation studies showed that the algorithm converges very slowly (7000 iterations). However, the algorithm converged very rapidly for values larger than stipulated by the proof.

There are several potential problems with the model used for this algorithm. The most serious is the assumption that the behavior of the underlying distributed system can adequately be predicted by a simple analytic formula. These algorithms require that the analytic function be continuous and differentiable. A second disadvantage is that termination detection requires global information. There is also room for improvement over this algorithm. First, it assumes that the accesses are uniformly distributed over the entire file resource. So, the algorithm does not detect and exploit any locality in the reference patterns. For example, processor P_i may exclusively access a *specific* fraction of the file resource. Kurose and Simha's algorithm only determines the *size* of the file resource fraction to allocate to P_i and does not consider its contents. Finally, there is a single copy of the file resource. Varying the number of copies of fractions of the file system may improve performance. For example, if some fraction x^0 is exclusively read, then a copy should be assigned to every processor. If fraction x^1 is only written, then there should be a single copy.

The data management economy of chapter 5 solves the above problems. First, no analytic model of the underlying system is assumed. Second, in the data management economy, all decision making is completely decentralized. Processor P_i only needs to know the read and write lease prices for a data object o_j, and the demand for access to o_j experienced locally to make decisions. We will show that the data management economy exploits locality in the access patterns to improve performance. The economy also adjusts the number of copies of a data object to reflect the ratio of read access to write access for the object.

The main disadvantage of the data management economy compared to the work of Kurose and Simha is that it is not possible to prove optimality of the allocation computed. This economy is evaluated through an implementation and simulation study.

2.2.2 *Job and Task Allocation*

Some previous work has applied economic concepts to the task, or job, allocation problem [68, 87]. In this problem, there is a set of jobs or tasks, denoted $T_1, T_2, \ldots, T_N$ that have to be executed in the distributed system. The problem is to assign the tasks to processors in the system. This assignment should fully utilize the processing power of each node in the system and must take into consideration any special dependencies a task has on hardware or software resources that are only available on a subset of the nodes. The object of the assignment is to optimize some performance metric such as task throughput, average task response time or total elapsed time.

If the set of tasks $\{T_1, T_2, \ldots, T_N\}$ is *static*, i.e. - no new tasks arrive, the problem is a pure task allocation problem. There may be some precedence relation between the tasks. It is usually assumed that the inter-task communication between each pair of tasks is known. So, the assignment of tasks to processors must also consider the communication cost incurred when T_i and T_j are assigned to different processors. In this section, we discuss the *Contract Net Protocol* [22, 83, 85]. This is a protocol for the task allocation problem that has been developed using a contract and negotiation metaphor. One of this protocol's sub functions is based on bidding, as is the load balancing economy of chapter 3.

If the set of tasks changes dynamically due to arrivals of new tasks and completion of others, the problem is modeled as a *load balancing problem*. The Contract Net Protocol has been used as a basis for the *Distributed Scheduling Protocol* of the *Enterprise System* [66, 67]. This protocol implements a distributed, dynamic load sharing algorithm and is surveyed in this section.

The idea underlying the Contract Net Protocol is the use of negotiation between intelligent, autonomous agents as a method for structuring distributed processing problems. In this model, the system as a whole tries to solve some single problem, e.g. - the *Distributed Sensing Problem (DSP)*[84] . In this problem, there is a set of processors and a set of sensing devices. The sensors collect information and report it to processing nodes, which analyze and interpret the information in cooperation with other processing and sensor nodes. The system as a whole attempts to compute the location and motion of objects in the area covered.

Complex problems such as the Distributed Sensing Problem require that the overall problem be broken down into separate sub problems. Each of these sub problems is a task which must be assigned to some node in the system. The Contract Net Protocol tries to solve a generalized task allocation problem.

The Contract Net Protocol starts when node P_i has a task T_j that is suitable for remote execution. P_i sends a **task announcement** message to other processors in the system. This message contains a description of T_j that can include estimated CPU time, memory requirements and any hardware dependencies. Figure 7 on page 33 depicts the phases of the Contract Net Protocol. In this example, P_1 is announcing task A and P_4 is announcing task B.

The recipient of a **task announcement** processes the message. This involves determining if the processor is able to perform the task, e.g. - contains the necessary hardware resources. This announcement is ranked using some problem specific criteria and is queued internally. When a processor P_j completes its current task and becomes idle, it picks the "best" locally queued announcement. P_j then submits a **bid** to the processor announcing the task (known as the *manager*). The **bid** contains a list of P_j's qualifications for performing the task. In the example of Figure 7 on page 33, processors P_1 and P_2 bid on task B and P_3 bids on task A.

The manager of an announced task waits specified time interval for bids to be submitted. After the interval expires, all submitted bids are evaluated and the best bid is determined. The best bidder is sent an **award message** telling it that it has won. This message contains any additional information needed for the winner to process the task. The losers are sent **cancel** messages informing them of their misfortune. In the Figure 7 on page 33, P_3 is awarded task A and P_1 is awarded task task B. Nodes P_1 and P_3 are part of a contract and may exchange information on the progress of task A .

The Contract Net Protocol is proposed as a paradigm for structuring large, complex problems in a distributed computing environment. In this protocol, all agents in the system are *cooperating* to achieve a *single goal*. In the economies in this dissertation, the agents *compete* for scarce resources and each agent attempts to meet its *individual goals*. The Contract Net Protocol performs resource

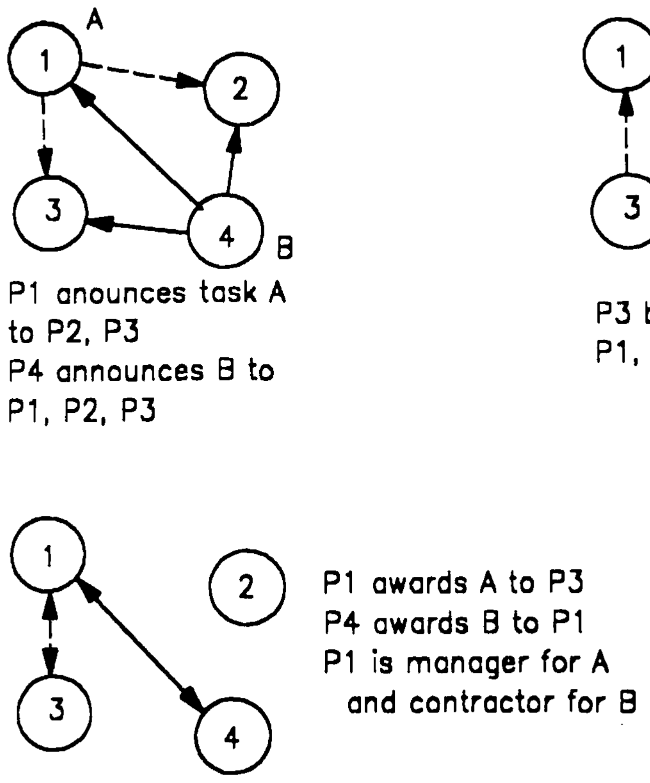

A
1
2
3
4
B
P1 anounces task A to P2, P3
P4 announces B to P1, P2, P3
P3 bids on A
P1, P2 bid on B
P1 awards A to P3
P4 awards B to P1
P1 is manager for A and contractor for B

Figure 7. The Contract Net Protocol: This figure depicts the Contract Net Protocol. Processor P_1 announces task A to P_2 and P_3. P_4 announces task B to all processors. P_3 submits a bid for A and P_1 and P_2 bid for B . Finally, P_1 wins B and P_3 wins A.

control in a distributed system for a different set of problems than our economies. Our economies operate in environments where physical and logical resources are scarce, and there is competition for these resources. The Contract Net Protocol is designed to match the logical demand for resources with the agents best capable of providing these resources. This protocol does not address environments in which there is competition for scarce resources.

The Contract Net Protocol has been used for the basis of the *Distributed Scheduling Protocol* of the *Enterprise System*. The Enterprise system is applied to the task allocation problem in a local area network of workstations. The goal is to utilize the capacity of idle nodes by remotely submitting tasks on the idle workstations. The Enterprise System has three layers. These are: 1) An interprocessor communication protocol, 2) A remote procedure layer, and 3) The *Distributed*

Scheduling Protocol (DSP). The first two layers provide the ability to remotely execute the tasks, and the DSP implements the algorithms that assign work to processors.

Using DSP, if processor P_i has a task T_j that is a candidate for remote execution, it sends a **task announcement** message to other workstations.[1] This message contains the priority of the task and requirements for executing it. The Enterprise System's goal is minimizing the average task response time. So, it attempts to implement *Shortest Job First* scheduling. The priority of task T_j is simply its estimated CPU service time, with lower service time implying higher priority. No other requirements are considered.

Each workstation receiving a **task announcement** ranks the announcement based on estimated service time and queues it. When the processor becomes idle, it submits a **bid** for the highest ranked announcement that is locally queued. The processor's estimated service time for the task, based on its CPU speed, is included in the bid. This is the processors qualification for executing the task.

Processor P_i waits for some time interval accumulating bids. When the interval expires, the task is awarded to the best bidder. The winner is the processor that reports the smallest estimated service time. An **award** message is sent to the winner, and the other bidders are sent **cancel** messages.

If processor P_i later receives a **bid** for task T_j that is "significantly better" than the bid that won, P_i can subsequently cancel the award and reassign the task to the better bidder. The definition of significantly better is a parameter to the system.

The Enterprise System and the Distributed Scheduling Protocol were evaluated through simulation. The main results that relate to the load balancing economy are the following:

1. Network size:
    a. Significant performance improvements were achieved for relatively small networks (8-10) nodes.
    b. Marginally decreasing performance improvements occur as network size increases.

[1] The Enterprise System uses different names for the messages sent. The names used in this section are of the equivalent message in the Contract Net Protocol.

2. Communication delay : The system improves performance when the communication delay of migrating a task is from 0 - 20% of the average CPU service demand.
3. Cancel/Reassignment : Allowing canceling and reassignment of tasks when better bids arrive is effective at low utilizations and counter productive at higher utilizations.
4. Error in estimates : The system is very insensitive to errors in the estimated service times of the jobs.

This dissertation's load balancing economy and the Enterprise System pursue fundamentally different approaches to performing resource allocation. In the load balancing economy, a job's priority is based on its initial endowment of money. We will show in chapter 3 that Shortest Job First as well as other scheduling policies can be implemented by this mechanism.

The load balancing economy is based on selfish optimization and competition. Processors solely try to maximize their individual revenue, and are not concerned with performance. The jobs compete with each other for the scarce communication and CPU resources. The Enterprise System is based on altruism, and each processor attempts to optimize the performance of the system as a whole.

The load balancing economy implements load balancing heuristics that the Enterprise System lacks. The Enterprise System works well over a wide range of CPU to communication ratios. It does not *adapt* its behavior to this ratio, however. The Enterprise System makes the same task allocation decisions regardless of the CPU/Communication ratio. We show in chapter 3 that the load balancing economy does modify its load balancing strategies based on the underlying communication and CPU processing rates. We will show how this economy can be tuned to change its load balancing strategy based on the job migration cost vs. the CPU service cost. Finally, the load balancing economy exhibits *flow control*. At high utilizations, poor jobs starve improving the performance of their relatively wealth competitors.

2.2.3 Flow Control

Some previous work has applied concepts from economics to the problem of *flow control* in computer communication networks. This previous work primarily deals with game theoretic models for Markovian networks [51]. At the end of this survey, the main differences between the previous work and the material of this dissertations are discussed.

In a computer communication network there is an implicit trade-off between throughput and delay. Throughput is the rate at which users transmit data through the network. Delay is the average elapsed time between the sending of the data and its arrival at the destination. Ideally, throughput should be "high" and delay "low." However, high throughput causes high delays, and decreasing delay constrains throughput. An equilibrium point between these two conflicting goals must be achieved.

Hsiao and Lazar [42, 43] observed the similarities between the flow control problem in a heterogeneous computer network and the theory of non-cooperatives games [82]. The two main similarities are:

1. Decentralization : Each user has an individual goal.
2. Competition : The users' goals conflict. For example, increasing the throughput of user A will increase the delay of another user B.

Hsiao and Lazar applied game theoretic concepts to describe the flow control problem for Markovian networks. In their model, there are K classes of users, and each class is represented by a player in the game. Player k acts as a controller allowing class k packets into the network. Player k's strategy for playing the game is to choose a vector

$$\lambda^k = (\lambda_1^k, \lambda_2^k, \dots, \lambda_{(N_k-1)}^k).$$

This strategy represents player k's choice of a window flow control policy [9]. λ_j^k is the rate at which player k allows packets into the network if there are already j class-k packets in the network that have not yet reached their destinations. User k's *window size* [9] is N_k. If $j = N_k$, player k does not allow any packets into the network.

Player k chooses a strategy that attempts to maximize his payoff. Hsiao and Lazar have player k attempt to maximize its average throughput subject to a delay constraint T^k. Let $\lambda = (\lambda^1, \lambda^2, \dots, \lambda^K)$ be the strategies chosen by the K players. Denote the average throughput achieved by user k under this set of strategies as $\gamma^k(\lambda)$, and let $\tau^k(\lambda)$ denote the average delay of class-k packets. Player k's payoff function $h^k(\lambda)$ is defined as

1. $h^k(\lambda) = \gamma^k(\lambda)$, if $\tau^k(\lambda) \leq T^k$.
2. $h^k(\lambda) = 0$, if $\tau^k(\lambda) > T^k$.

Simply stated, player k wants to maximize throughput as long as the average delay is less than T^k. However, if the delay constraint cannot be met, player k 's payoff is 0. This payoff function accurately models a network using timeout based error detection and transmission. If the packets are not delivered within the delay constraint, the effective throughput will be 0 due to the timeout error detection.

Hsiao and Lazar prove that player k 's optimal strategy λ^k for any choice of strategies of the other players is a window policy. In this policy, player k allows no packets into the network if there are L_k or more packets in the system. The value of L_k can be derived as the results of a Linear Program.

Since multiple players are individually trying to optimize different functions, Hsiao and Lazar propose *Nash Equilibrium* [82] as the definition of the set of optimal player strategies. Simply, a set of strategies

$$\lambda^* = (\lambda^1, \lambda^2, \ldots, \lambda^K)$$

is a Nash Equilibrium if no player can achieve a higher payoff (i.e. - More throughput without violating delay constraints) by unilaterally changing its strategy. Hsiao and Lazar then derive the set of flow control problems for which the Nash Equilibrium exists.

Bovopoulos and Lazar [11] expand on the work of Hsiao and Lazar by presenting algorithms for *computing* a Nash Equilibrium under two different network models. In the first model, each player has the payoff function described above. There are only two players in the game, however. The first algorithm is as follows:

Step 1: Start with an initially empty network, with $\lambda^1 = \lambda^2 = 0$.

Step 2: Player 1 chooses λ^1 that maximizes his average throughput γ^1 (given λ^2) without violating the delay constraint $\tau^1 \leq T^1$.

Step 3: Player 2 may not be able to allow any traffic into the system without having $\tau^2 > T^2$. If this is the case, the game is over. If not, player 2 chooses λ^2 that maximizes γ^2 (given λ^1) with $\tau^2 \leq T^2$.

Step 3: If $\tau^1 \leq T^1$, the game is over. Otherwise, go to step 2.

Bovopoulos and Lazar prove that this algorithm converges to the Nash Equilibrium.

In the second model, there are K players, and each player uses *power* as its payoff function. Player k chooses a weighting factor β^k and attempts to maximize

$$\frac{(\gamma^k)^{\beta^k}}{\tau^k}.$$

There is a single exponential server with rate μ and each player's controls are state independent, i.e. $\cdot\ \lambda_j^k = \lambda_i^k$. Given this, player k 's power is simply

$$(\gamma^k)^{\beta_k} \bullet (\mu - \sum_{i=1}^{K} \lambda^i).$$

Bovopoulos and Lazar show that the Nash Equilibrium can be computed as the fixed point of linear system $x = Bx + b$. In this formulation, $x^T = [\lambda^1, \ldots, \lambda^K]$, $b^T = [\mu\beta_1, \mu\beta_2, \ldots, \mu\beta_K]$, and B is a $K \times K$ matrix with:

1. $b_{ii} = 0.$
2. $b_{ij} = \dfrac{\beta^i}{1 + \beta^i},\ i \neq j$.

They then present an asynchronous algorithm based on chaotic iteration [16] that computes the fixed point x. This algorithm is composed of updating the x_k's at discrete time points $t = 0, 1, \ldots$. At point t, the value x_k at the next time point, denoted $x_k(t+1)$, is either the value at time $t, x_k(t)$, or is updated using constant values from the matrix B and the vector b , and values of $x_i(t - t')$ for the other players. They then prove that this algorithm converges provided that the delay in propagating information from player i to player k is bounded, and if the weights β^k satisfy certain constraints.

Douglieris and Mazumdar [70] also studied a Nash Equilibrium in a single server queueing system with K users. They also used power as the players' payoff functions, but they expanded the set of weights $\beta^1, \beta^2, \dots , \beta^K$ for which the chaotic iteration algorithm is proven to converge. Douglieris and Mazumdar proposed an iterative, synchronous algorithm similar to the first algorithm of Bovopoulos and Lazar for computing the Nash Equilibrium in this model. Let $P_i(\lambda^1, \lambda^2, \dots , \lambda^K)$ be the power of player i, $i = 1, 2, \dots , K$, and let λ^i_k be player i's chosen throughput in iteration k. The following updates are performed in the k-th iteration:

1. Player 1 chooses $\lambda^1_{(k+1)}$ as the λ maximizing $P_1(\lambda, \lambda^2_k, \dots , \lambda^K_k)$.
2. Player 2 chooses $\lambda^2_{(k+1)}$ as the λ maximizing $P_2(\lambda^1_{(K+1)}, \lambda, \lambda^3_k, \dots , \lambda^K_k)$.
3. Player i chooses $\lambda^i_{(k+1)}$ as the λ maximizing $P_i(\lambda^1_{(k+1)}, \lambda^2_{(k+1)}, \dots , \lambda^{(i-1)}_{(k+1)}, \lambda, \lambda^{(i+1)}_k, \dots , \lambda^K_k)$.
4. Player K chooses $\lambda^K_{(k+1)}$ as the λ maximizing $P_K(\lambda^1_{(k+1)}, \dots , \lambda^{(K-1)}_k, \lambda)$.

If this algorithm converges, it computes a Nash Equilibrium [63]. Finally, they prove that their algorithm converges for $K = 2,3$, or if $\beta^1 = \cdots = \beta^k = \beta$.

Dougliers and Mazumdar [69] examine *Pareto-optimality* in a single server queuing system in which each user attempts to maximize power. Let

$$P_i = (\gamma^i)^{\beta_i} \bullet (\mu - \sum_{i=1}^{K} \lambda^i)$$

be the power of player i. This set of throughputs is defined as Pareto-optimal if for any other set of throughput λ^i_0, $i = 1, 2, \dots , K$:

1. For all i, $P_i(\lambda^i) = P_i(\lambda^i_0)$, or
2. For at least one j, $P_j(\lambda^j) > P_i(\lambda^i_0)$.

This simply states that $\lambda^1, \lambda^2, \ldots, \lambda^K$ is Pareto-optimal if no one can be given a higher power without forcing a lower power on some other player. They present a formula defining necessary and sufficient conditions for a set of λ^i to be Pareto-optimal. Finally, they show that in special cases, the Pareto-optimal throughputs also maximize the weighted sum of the players' powers. The weighted sum of powers has been defined as *system power* [10, 48] and has been studied in non-economic contexts.

The flow control economy uses Pareto-optimality as the definition of an optimal allocation of resources to the agents in the system. Pareto-optimality is a stronger definition for an optimal allocation of resources than the Nash Equilibrium. A Nash Equilibrium is only optimal in the sense that no single agent can unilaterally take an action that improves upon its allocation of resources. If agents are allowed to communicate or cooperate, it is possible to compute an allocation of resources that is as good for all agents, and strictly better for some.

Secondly, the previous work on both Nash Equilibria and Pareto optimality can compute resource allocations in which some agents receive no resources. That is, some player i may receive throughput equal to 0. In addition to computing an optimal throughput-delay trade-off, flow control algorithms try to guarantee that all agents receive a "fair" allocation of resources. Through the explicit use of money allocated to agents with which they purchase resources, every VC receives a fair share of resources in the flow control economy. In chapter 4, we compare the fairness properties of the flow control economy with Nash Equilibria.

The flow control economy is applied to a different network model than the previous work. This difference has two aspects. First, each agent i has a throughput goal γ^i that it tries to achieve. If this can be done, agent i tries to minimize the delay of achieving this throughput. The throughput goal is not a constraint, however, and the agent can settle for less throughput. In some sense, this is the dual to the player payoff functions in the above games.

Another difference is that the flow control economy is based on a *virtual circuit* computer network. In such a network, each agent has a predefined path connecting its *source* node to it single *destination* node. In practice, most networks are based on the virtual circuit model [9]. So, in this sense the flow control economy is based on a more realistic network model.

2.2.4 Multiple Access Protocols

The problem of designing optimal multiple access protocols for broadcast communication networks [81] has been studied using concepts and algorithms from economics. In this problem, there is a broadcast communication medium being shared by multiple independent network nodes. The ethernet [71] and packet radio networks [50] are examples of such networks. Each node or station in the network has data packets generated by some source that it wishes to transmit over the network. The communication medium is divided into fixed length slots. To transmit a packet, node N_i waits until the next slot starts and then broadcasts the packet. If no other node N_j attempts to transmit a packet during this slot, the packet transmitted by N_i is correctly received at its destination. If, however, N_j also attempts to transmit a packet during the slot used by N_i, a *collision* occurs and neither packet is successfully transmitted.

Protocols for the multiple access problem assign a *transmission probability* p_i to each node N_i. If node N_i has a packet it wishes to transmit in the next available slot, it broadcasts this packet with probability p_i . If N_i does not send the packet during the slot, or there is a collision, another aspect of the protocol determines the next slot to try. Figure 8 on page 42 depicts packets, slots and collisions in a broadcast network. The problem examined in this section is computing the transmission probabilities p_i for each node N_i that optimizes throughput.

Many multiple access protocols have been proposed [56], and different protocols are optimal for different traffic arrival patterns and assumptions. Yemini and Kleinrock [89, 90] showed that previously known protocols that are optimal for differing assumptions are actually special cases of *Pareto-optimality*.

Yemini and Kleinrock showed that optimal protocols achieve a balance between throughput and silence. If node N_i has a packet to transmit at slot s_j and does not, its throughput is diminished but the throughput of other nodes benefits from the fact that they will not collide with N_i during slot s_j. Let E_i be node N_i's lost throughput due to choosing silence, and let S_j be the attained throughput of node N_j, for $j = 1, 2, \ldots, K$. These terms are functions of the individual transmission probabilities $p_1, p_2, \ldots, p_K$. For every "optimal" choice of transmission probabilities, there exists a set of prices $c_1, c_2, \ldots, c_K$ for which

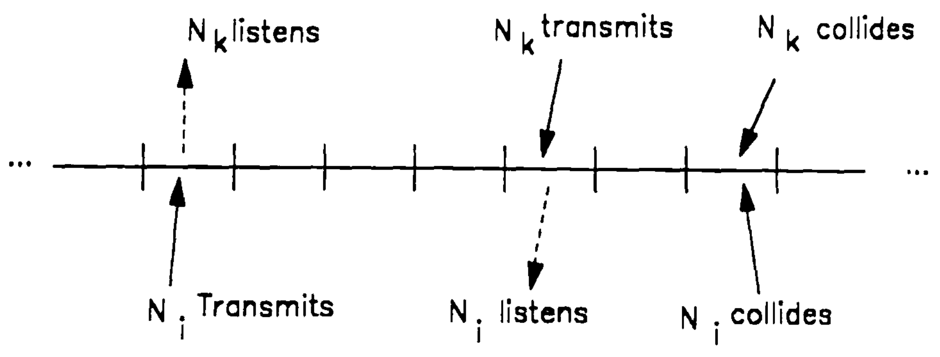


Figure 8. Multiple Access Problem

$$c_iE_i = \sum_{j=1,j\neq i}^{j=K} c_jS_j,\ 1 \leq i \leq K,$$

for all i. The term c_iE_i is the dollar value of N_i's lost throughput, and

$$\sum_{j=1,j\neq i}^{j=K} c_jS_j$$

is the dollar value of the throughput attained by other nodes when N_i is silent. For a choice of probabilities to be optimal, the dollar value of silence must equal the dollar value of total throughput. Yemini and Kleinrock then show that various values of the c_j correspond to previously known optimal protocols.

Kurose, Yemini and Schwartz [55, 57] present a decentralized algorithm for computing the prices that define the optimal transmission probabilities. Instead of attempting to optimize a common system wide goal, each node attempts to maximize its individual throughput.

This algorithm is built on the concept of a *fictitious resource*. These resources capture the interaction between node N_i's silence and N_j's throughput. For each pair of nodes (i, j), there are two fictitious communication resources $x_{i\to j}$ and $x_{j\to i}$. These resources reflect the demanded transmission probabilities of the node and the silence demanded of other nodes. For example,

1. $x^i_{i\to j}$ is N_i's demand to N_j that N_i be allocated transmission probability $x^i_{i\to j}$.
2. Conversely, $x^j_{i\to j}$ is node N_j 's demand that N_i remain silent with probability $x^j_{i\to j}$.

The demands are similarly defined for $x^j_{j\to i}$ and $x^i_{j\to i}$. Since the fictitious resources are transmission probabilities, $x^i_{i\to j}$ and $x^j_{i\to j}$ are in the interval $[0,1]$. Furthermore, since $x^i_{i\to j}$ is N_i's demand to transmit with respect to its competitor N_j and $x^j_{i\to j}$ is competitor N_j 's demand that N_i remain silent, these are only consistent if

$$x^i_{i\to j} + x^j_{i\to j} = 1.$$

The economic model used for this algorithm is a pure *Exchange Economy* [40]. In this model, the agents are not allocated money for purchasing resources. Instead, the supply of the resources is assumed to be initially distributed among the agents in the economy. The agents then "trade" their initial resource allocations for ones they consider better. The initial allocation of resources owned by node N_i is called its ***endowment***. In this economy, node N_i is allocated quantities of the two resources that relate to its interaction with each other node N_j. These are $x^i_{i\to j}$ and $x^i_{j\to i}$. Node N_i's endowment of these resources is denoted

$$\bar{x}^i_{i\to j}, \text{ and } \bar{x}^i_{j\to i}.$$

For each fictitious resource $x^i_{i\to j}$ there is an associated price $p_{i\to j}$. These prices define the relative value of the fictitious resources and determine the value of each node N_i's endowment. The value of the endowment is the revenue N_i could earn by selling its resources at the given prices. At any set of prices

$$\vec{p} = \{p_{i\to j} \mid 1 \leq i, j \leq K \text{ and } i \neq j\}$$

the wealth of node N_i is

$$W_i = \sum_{i=1,\, i\neq j}^{K} [p_{i\to j}\bar{x}^i_{i\to j} + p_{j\to i}\bar{x}^i_{j\to i}].$$

At a set of prices $\vec{p}$, each node demands the allocation of fictitious resources that maximizes its throughput subject to its budget constraints. The cost of the resources demanded by N_i at prices $\vec{p}$ cannot exceed the value of N_i 's initial allocation of resources *at the current prices* $\vec{p}$. Formally, if $\{\tilde{x}^i_{i\to j}, \tilde{x}^i_{j\to i} \mid \forall (j \neq i)\}$ is N_i's demand at $\vec{p}$, then

$$\sum_{i=1,\, i\neq j}^{K} [p_{i\to j}\tilde{x}^i_{i\to j} + p_{j\to i}\tilde{x}^i_{j\to i}] = \sum_{i=1,\, i\neq j}^{K} [p_{i\to j}\bar{x}^i_{i\to j} + p_{j\to i}\bar{x}^i_{j\to i}].$$

The algorithm for computing the optimal set of fictitious resources depends on the specific multiple access protocol. Kurose, Yemini and Schwartz give algorithms for the Slotted Aloha Protocol [2, 3] and for a time window protocol [55].

Since the resources represent probabilities, the demands are consistent only if

$$x^i_{i\to j} + x^j_{i\to j} = 1$$

for all i, j. If,

$$x^i_{i\to j} + x^j_{i\to j} > 1$$

resource $x_{i\to j}$ is under priced, and $p_{i\to j}$ should be increased. If, however,

$$x^i_{i\to j} + x^j_{i\to j} < 1$$

the price $p_{i\to j}$ is too low and should be increased. In their algorithm, Kurose, Schwartz and Yemini use a tatonement process to update prices. The update is defined by the following formula:

$$p_{i\to j} = p_{i\to j} + c_{i\to j}[1 - x^i_{i\to j} - x^j_{i\to j}],$$

where $c_{i\to j}$ is a *step size* constant.

Kurose, Yemini and Schwartz proved that if their economy computed an equilibrium in which

$$x^i_{i\to j} + x^j_{i\to j} = 1$$

for all i, j, then the allocation was Pareto-optimal. Proving that an equilibrium price vector exists, and that this algorithm converges are extremely difficult problems. Kurose, Yemini and Schwartz show that the demand of the nodes in this network are not continuous, and standard techniques from economics cannot be applied to prove existence of equilibrium.

Kurose, Yemini and Schwartz implemented and simulated their economy for a 4 node network. These simulations demonstrated empirically that the tatonement process converged to an equilibrium. They also showed that their economy rediscovered previously known optimal transmission probabilities for the Slotted Aloha Protocol. Finally, they examined the effectiveness of the initial endowment of resources to agents as a priority mechanism. The simulations revealed that increasing the initial endowment did increase the equilibrium throughput of an agent.

The main disadvantage of this work is complexity. For a K node system, there is one physical resource (communication medium) but there are $K^2 - K$ fictitious resources that the economy must allocate.

3.0 The Load Balancing Economy

This chapter presents an economy that uses decentralized competition and a price system to perform load balancing. The load balancing problem in a distributed system is to allocate CPU and communication resources to jobs entering the system to minimize the average response time [29].

The load balancing economy is implemented and tested on the *Network Simulation Testbed* (NEST) [25]. As with previous dynamic load balancing algorithms, the load balancing economy cannot be evaluated analytically [29]. A simulation study is presented to evaluate the economy's performance. The implementation and simulation study also allow us to evaluate the software structure of the economy, and determine its complexity.

The main result of this section is a set of experiments that shows that the economy substantially improves performance for a wide choice of system configurations and agent behavior rules. We compare the economy to a representative, non-economic load balancing algorithm, and experiments demonstrate that the economy achieves better performance. Experiments also show that the economy implements a broad spectrum of load balancing strategies and can adapt its strategy to the relative power of CPU vs. communication. Finally, experiments evaluate the effects of the price system on individual jobs. These are: 1) Flow control of poor jobs, 2) Job wealth as a priority mechanism.

Section 1 defines the load balancing problem and the model of the underlying distributed system. Section 2 describes the load balancing economy and presents a running example. The main results of this chapter are contained in section 3. This section presents the results of several experiments that measure the performance of the economy and compare it to a non-economic load balancing algorithm. Section 4 is a comparative survey of previous work on load balancing. Section 5 contains a conclusion and discusses directions for future work. There are two appendices to this chapter. The first is a discussion of some the issues raised in this chapter. The second is a detailed description of the non-economic load balancing algorithm used for the comparisons in section 3.

3.1 Problem Statement and System Model

The distributed system consists of N processing nodes $P_1, \dots, P_N$. The CPU power of node P_i is defined by a processing speed parameter r_i. A job that requires μ CPU seconds on an idealized processor with $r = 1$, requires $\frac{\mu}{r_i}$ seconds on processor P_i. The processors are connected by a point-to-point network defined by an edge set $E = \{e_{i,j}\}$. Edge $e_{i,j}$ represents a unidirectional communication link connecting processors P_i and P_j. This link has a delay of d_{ij} seconds per byte.

Jobs originate at all processing nodes and seek CPU time on any processor. A job must leave the system at its original node. The service requirements of a job are defined by

1. μ - The CPU service time of the job on an idealized processor P_i with $r_i = 1$.
2. ReqBC - The number of bytes needed to describe the job. Each time a job migrates from one processor to another seeking CPU service, ReqBC bytes must be sent.
3. RspBC - The number of bytes needed to describe the result of executing the job.

The assumption that the resource demands of every job are known when it enters the system is not realistic for all applications. There may be statistical information based on previous submissions of the job, however. The system model of this chapter is an idealization of this case. Adding behavioral rules that allow agents to deal with incomplete knowledge is left for future work.

The load balancing problem is to design an algorithm that minimizes mean job waiting time by migrating jobs to balance the workloads of the processor nodes. The algorithm should be effective over a broad range of distributed system configurations.

3.2 The Economy

The load balancing economy regulates the sharing of CPU time and communication capacity. Each job J is provided with an initial allocation of money m_J with which it tries to purchase $\frac{\mu_J}{r_i}$ CPU seconds on some processor P_i. The policy for assigning the initial endowment is discussed later. The job is allowed to migrate through the system in search of CPU service, but it must pay each time it crosses a link.

Processor nodes sell CPU time and communication bandwidth to the jobs. Processor node P_i independently sets the prices it charges for its local CPU and each link $e_{i,j}$. P_i's goal is to maximize revenue. Each processor node P_i advertises its resource prices in bulletin boards at P_i and adjacent nodes.

3.2.1 *The Jobs*

A job performs three operations to purchase resources. These are: (1) Compute a *Budget Set*, (2) Apply a *Preference Relation* to the elements of the budget set and (3) Generate a *Bid* for the most preferred budget set element.

We will use a running example to illustrate job behavior. Figure 9 on page 49 depicts a small distributed system. There are two jobs at processor P_1. J_1 has $\mu = 30$ ms and J_2 has $\mu = 10$ ms. Both jobs have been allocated \$25 to purchase their resources and both have $ReqBC = RspBC = 1000\ bytes$. The dollar figures next to each node and data link represent the information in P_1's Bulletin Board. CPU time at P_1 is selling for \$1.00 per millisecond. The last advertised CPU prices at P_2 and P_3 are \$0.10 per ms and \$0.75 per ms respectively. The current price P_1 is charging to get to P_2 is \$0.001 per byte and the cost to get to P_3 is \$0.002 per byte. How P_1 sets these prices is described below. In this figure, all processors have a CPU rate of 1 and each data link has capacity 1 Mb/second.

3.2.1.1 **Compute the Budget Set**

Let m_J be job J's wealth. A budget set element is an ordered pair (k, C_k) with $C_k \leq m_J$. C_k is the estimated cost of servicing job J at P_k. C_k is composed of:

1. The cost of buying $\frac{\mu_J}{r_k}$ CPU seconds at P_k.
2. The cost to cross the link to get to P_k.
3. The cost to get from P_k to the processor at which J entered the system.

The local bulletin board contains the price information used to compute C_k. Prices for local resources are exact, but the prices for remote resources are only estimates. These prices may change as J migrates through the network.

In our example, the budget sets for J_1 and J_2 are:

- $B_1 = \{(2,\$5.00)\}$.
- $B_2 = \{(1,\$10),(2,\$3),(3,\$11.50)\}$.

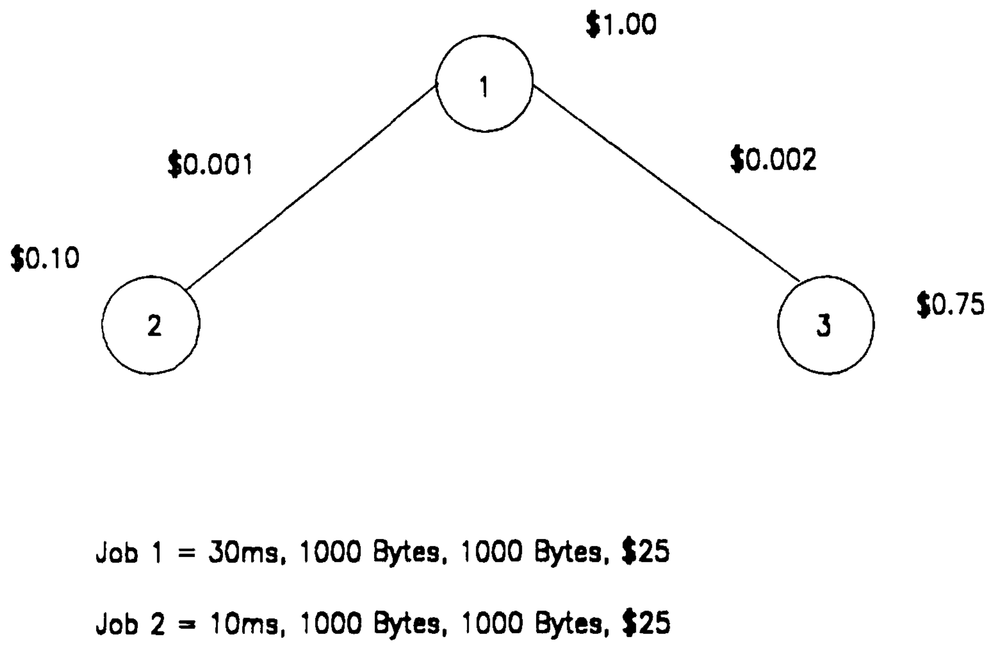

$1.00
1
$0.001
$0.002
$0.10
2
3
$0.75
Job 1 = 30ms, 1000 Bytes, 1000 Bytes, $25
Job 2 = 10ms, 1000 Bytes, 1000 Bytes, $25

Figure 9. A Simple Distributed System: There are three processors and two jobs in the system. Dollar values next to processors and links represent the bulletin board information available at processor 1.

3.2.1.2 Computing Preferences

A job J must decide which element of its budget set is "best." To do so, the job uses a *Preference Relation* that imposes a partial ordering on the elements of the budget set. This relation is denoted $\succcurlyeq_J$, and $(j, C_j) \succcurlyeq_J (k, C_k)$, if J prefers service at P_j at cost C_j at least as much as service at P_k at cost C_k. We consider three preference relations:

1. **Price** - The job wants to be serviced as cheaply as possible. In our example, both J_1 and J_2 prefer service at P_2.
2. **Service Time** - A job prefers the element of the budget set at which it can be most quickly serviced. The estimated service time at P_k is the CPU service time at P_k plus the link delay of migrating to P_k and returning. If J is currently at P_i, the service time of neighbor P_k is

$$\frac{\mu_J}{r_k} + ReqBC_J \bullet d_{lk} + RspBC_J \bullet d_{kl}. \quad [3.1]$$

In our example, d_{12} and d_{21} are 8×10^{-6} seconds per byte i.e. - 1Mb/s links, and the service time of J_2 at P_2 is

$$26ms = \frac{10ms}{1} + 1000(8 \times 10^{-6})s + 1000(8 \times 10^{-6})s$$

3. **Service Time vs. Price** (STVP) - In the STVP preference relation, both service time and price are considered. Under this preference, $(k, C_k) \succcurlyeq_J (j, C_j)$ if $C_k + A \bullet ST_k \leq C_j + A \bullet ST_j$, where 1) ST_k is the service time at P_k given by formula [3.1], and 2) A is a constant that defines the relative importance of service time versus cost. In our example if $A = 1$, J_2 prefers service at P_1, and if $A = 0.1$, J_2 prefers service at P_2 .

3.2.1.3 Bid Generation

After computing the budget set and applying its preference relation, job J knows where it wants to be serviced. The job now submits a bid for the next resource it needs. The next resource is either the local CPU or a link to an adjacent node.

If J has m_J dollars remaining and the optimal element of the budget set is (k, C_k) , then J's estimated surplus is $m_J - C_k$. In our example, J_1's estimated surplus at P_2 is \$20.

Job J bids the current price plus a fraction of its surplus defined by a parameter S_J. In our example, the price for the link to P_2 is \$0.001, and J_1 's surplus is \$20. So, J_1 will bid

$$\$0.001 \bullet ReqBC_1 + (\$25 - \$5) \bullet S_1$$

for link e_{12}. The effects of the parameter S_J are studied in the experiments discussed in section 3.3.5.

3.2.2 Processors

A processor node P_i performs three functions:

1. *Auction Resources* : When a resource (link or CPU) becomes idle, P_i holds an auction to determine which of the jobs gets the resource next.

2. *Advertise* : Based on a local price change, P_i may send an advertisement message to its neighbors.
3. *Update Prices* : Using arriving advertisements and the results of local auctions, P_i updates prices in the local bulletin board.

3.2.2.1 Auction Resources

We studied two auction models for the economy. They are: 1) Sealed Bid and 2) A hybrid of Dutch and English Auctions. Sealed Bid, Dutch and English Auctions auctions were described in chapter 2. The hybrid model tries to find the highest price that can be charged for a resource by increasing the asking price when a bid is submitted (English), and decreasing the asking price when no bid is submitted (Dutch). In both models, a job J is allowed to overbid using its surplus and S_J . This allows strategy to play a role in the auction process. The effect of strategy is discussed later.

3.2.2.2 Price Updates

When a processor sells a resource to a job, the resource's entry in the bulletin board is set to the unit price of the sale. In our example, if $S_2 = 0.33$ and the Service Time preference is used, J_2 will bid

$$\$15.00 = \$1.00 \cdot \mu_2 + 0.33 \cdot (\$25 - \$10)$$

for P_1's CPU, and if $S_1 = 0.25$, J_1 will bid

$$\$6.00 = \$0.001 \cdot ReqBC + 0.25 \cdot (\$25 - \$5)$$

for link e_{12}. If these are the winning bids, P_1 sets its CPU price to $\$1.50 = \$15/10$ and the price for link e_{12} to $\$0.006 = \$6.00/1000$.

3.2.2.3 Advertising

Each time the CPU price changes at a processor, the processor broadcasts the new price to all of its neighbors. In our example, P_2 and P_3 are told of P_1's new CPU price of \$1.50 per millisecond. They insert this new value in their bulletin boards so that jobs at these processors can make decisions regarding service at P_1. The overhead of the advertisement policy and alternate policies are discussed in the appendix.

3.3 Performance Experiments

We implemented the behavioral rules for the processors and jobs on Columbia University's *Network Simulation Testbed (NEST)* [25]. We then simulated the system to measure the performance and behavior of the economy. The distributed system we simulated was a nine processor mesh (Figure 10 on page 53). The processing rate of each processor was 1.0, i.e. - a homogeneous system, and each communication link had a capacity of 1 Mb/s.

In the simulation, jobs arrive at all processors in the system. The jobs' service times were drawn from an exponential distribution with a mean service time of 30 ms. The arrival process was assumed to be Poisson with the same rate at all processors. The mean inter-arrival time was varied to generate system utilizations from 0 to 90 percent. We also assumed that both the ReqBC and RspBC parameters were the same for all jobs and fixed at 1000 bytes. Except for the experiments of section 3.3.5, all jobs were allocated $5.

If all jobs use the same preference relation and all processors use the same auction model, the economy can be classified using the preference and auction names. For example, an economy in which all jobs use the Price preference and all processors use Sealed Bid auctions can be classified as a Price-Sealed Bid or as a Sealed Bid-Price economy. This classification will be used to define the economies discussed in this section.

3.3.1 Job Waiting Time

The main performance metric we studied was the mean job waiting time (response time - CPU service demand). Figure 11 on page 54 plots the mean job waiting time versus utilization for the Sealed Bid-Price, Sealed Bid-Service Time and Sealed Bid-STVP economies. These waiting times are compared to the case of no load balancing (an M/M/1 queueing system using shortest-job-first scheduling (SJF)) and the case of theoretically optimal load balancing (an M/M/9 system using SJF scheduling). The M/M/9 case is depicted for comparison purposes only. The M/M/9 system has no communication delays and all information is globally available and exact. No load balancing

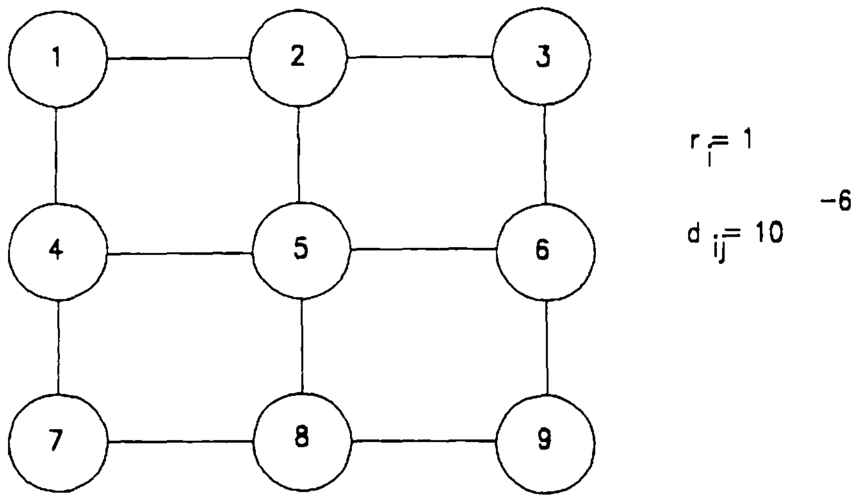


Figure 10. A simple distributed system

algorithm can achieve the same level of performance. Figure 12 on page 55 shows the job waiting times yielded by the three job preference relations when the processors use the Hybrid auction model to allocate the resources.

These figures demonstrate that the load balancing economies substantially improve performance compared to no load balancing. The decrease in mean waiting time is greater than 60% for all economies at high utilizations. A second observation that can be made is that there is little difference between the performance levels achieved by the three preference relations. Under the Sealed Bid model, Service is marginally better than the other two preferences. When the processors use Hybrid auctions, Price is the best. The explanation for the relative ordering of preference relations' performance are discussed in experiments in the following sections.

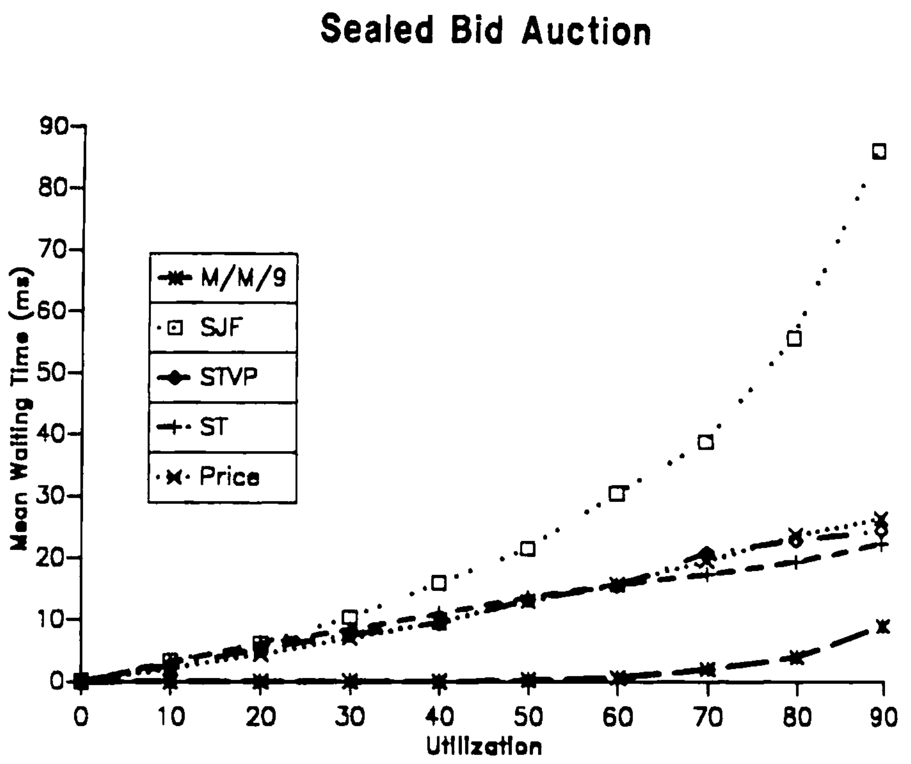


Figure 11. Job Waiting Times: This graph plots mean job waiting time vs. system utilization for the three job preference relations when processors use Sealed Bid auctions. The no load balancing (SJF) and optimal load balancing (MM9) boundary cases are included for comparison. The best economy is one in which all jobs use the Service (ST) preference. All jobs using the Price preference is marginally the worst.

We compared the Load Balancing Economy to Livny's HOP 1 algorithm [64]. We chose this algorithm because of its general applicability. No heuristics specific to a particular system model are used. To make the comparison fair, we modified the HOP 1 algorithm so that it uses the available information about the service times of the jobs. In its original version, the algorithm assumes that the CPU service times are not known. A complete description of the modified HOP 1 algorithm is given in an appendix.

We simulated and tuned the HOP 1 algorithm to optimize its performance for the distributed system used by the load balancing economy. Figure 13 on page 56 plots the mean job waiting time of the HOP 1 algorithm as a function of system utilization. This graph includes the mean job waiting times of the no load sharing and optimal load sharing cases for comparison purposes. The

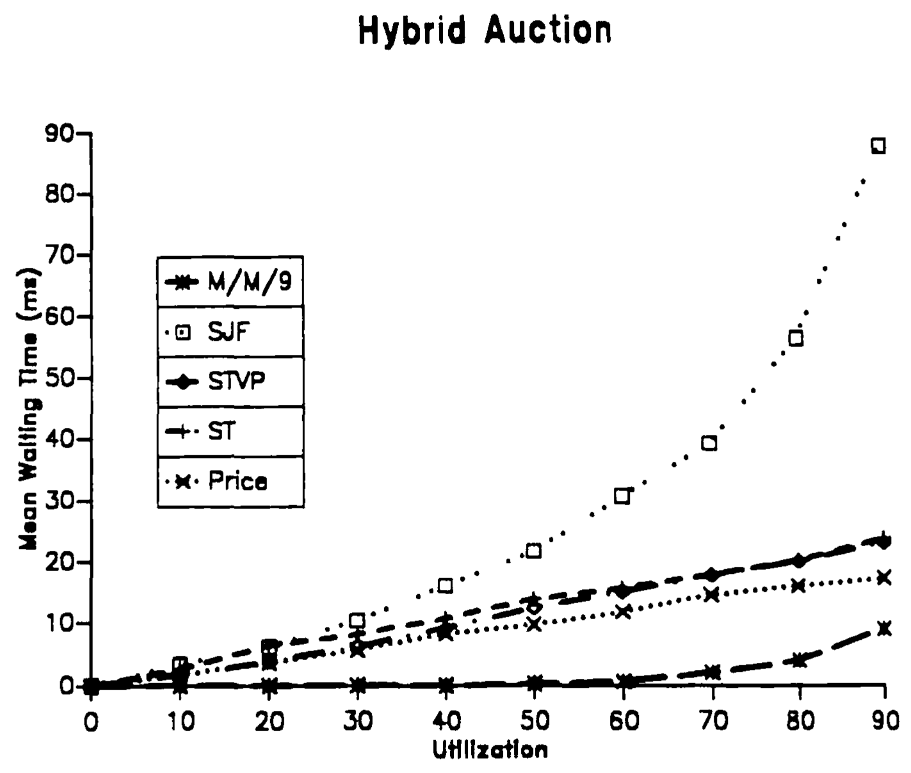


Figure 12. Job Waiting Times: This graph plots mean job waiting time vs. system utilization for the three job preference relations when processors use the Hybrid auction model. The no load balancing (SJF) and optimal load balancing (MM9) boundary cases are include for comparison. The best economy is one in which all jobs use the Price preference. The worst performance occurs when all jobs use the Service preference.

two best curves in this figure represent the waiting times achieved by the best preference relations when a Sealed Bid and a Hybrid auction model are used by the processors. These preferences are Service Time and Price respectively. The Hybrid Auction-Price economy achieves better performance than the than the HOP 1 algorithm for all utilizations. The Sealed Bid-Service economy is as good as HOP 1 at low utilizations, and better at higher utilizations.

This study shows that for the load balancing problem, competitive economic concepts can achieve levels of performance that are as good as traditional cooperative algorithms. In some cases, the performance achieved is actually better than a non-economic algorithm.

3.3.2 Spectrum of Load Balancing Strategies

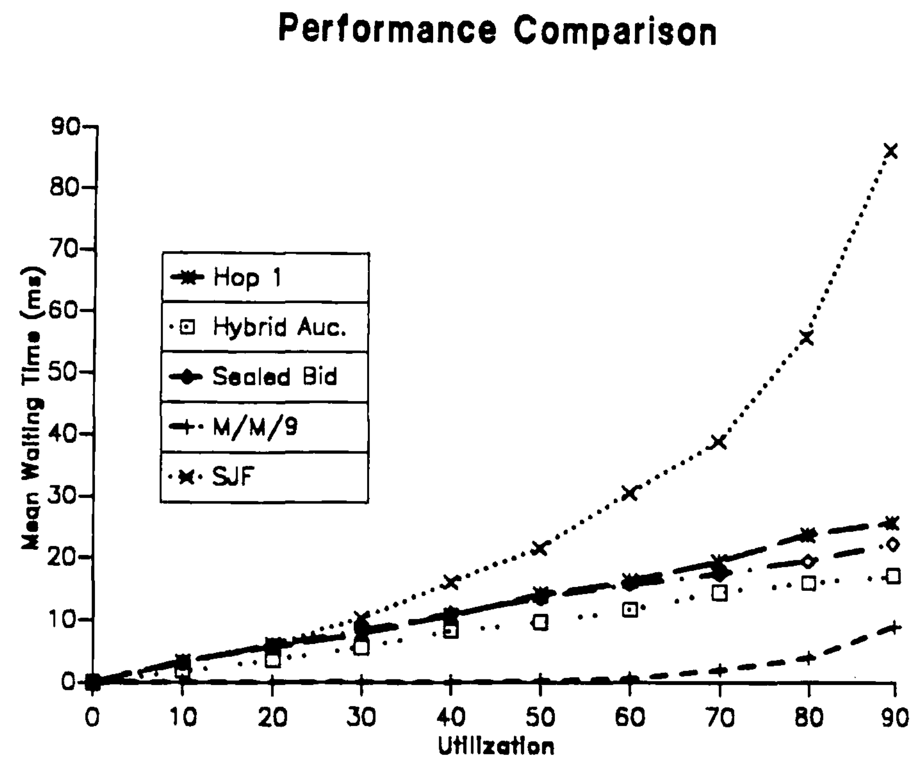


Figure 13. Economic vs. Non-Economic Algorithms: This graph plots the mean job waiting times of the HOP 1 algorithm (HOP 1), the best economy under Sealed Bid auctions (Service Time preference) and the best economy under Hybrid auctions (Price).

In the experiments of the preceding section, the choice of a preference relation for the jobs had a minor effect on performance. However, the *way* in which the economies balance the loads is remarkably different. Figure 14 on page 57 depicts the mean job *migration distance* for the three preferences when a Sealed Bid auction is used. The migration distance is the number of links a job crosses before receiving CPU service on some processor. Figure 15 on page 58 shows the mean migration distance under the Hybrid auction model. In both cases, the migration distance of the Price preference is three times that of the Service Time preference. The STVP preference causes jobs to migrate more than the Service Time preference and less than Price for both auction models.

The reason for this behavior is as follows. In a mesh topology, each processor has at least two neighbors. The current price for CPU time at a processor is a random variable and the probability of processor P_i having at least one neighbor charging less for CPU time can be large. When the

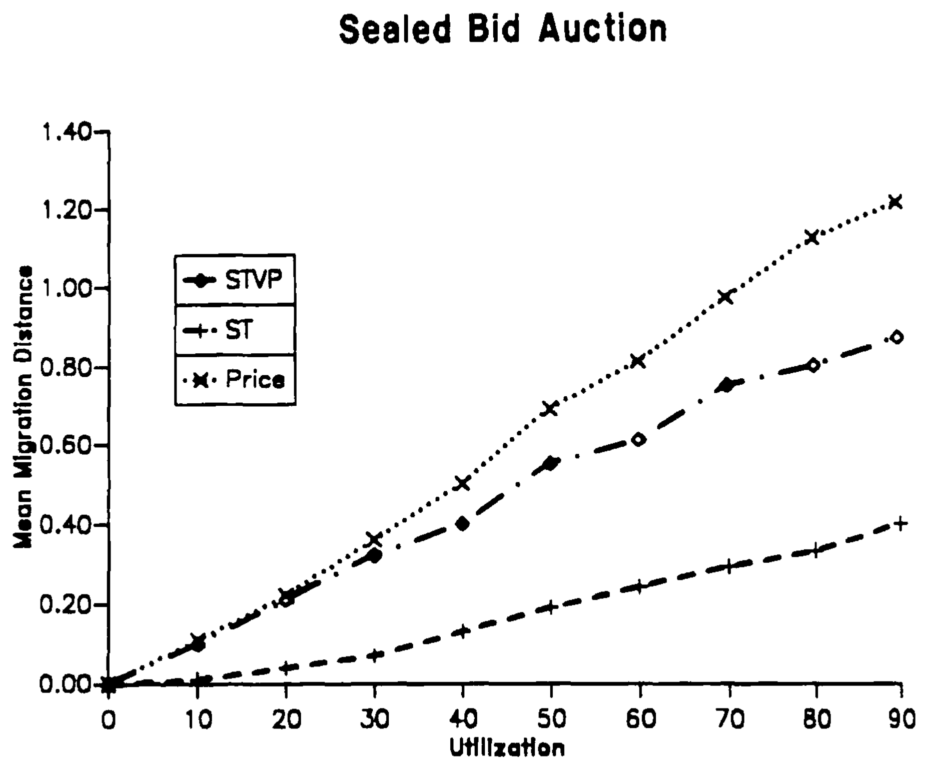


Figure 14. Job Migration Distance: This graph plots the mean distance a job migrates before receiving CPU service for each of the three preferences under a Sealed Bid auction model.

Price preference is used by the jobs, all jobs at P_1 attempt to migrate to a cheaper processor. This phenomenon causes the high migration rates in the system where the Price preference is used. The system does not become unstable, however, because the high migration rates cause the price for communication bandwidth to increase and migration becomes less attractive. This is especially true in the Hybrid auction economies. At high utilizations, the migration rates taper off, and are approximately 40%-50% lower than in the Sealed Bid economies.

When migration is attractive, the jobs compete for the communication links. Shorter jobs have smaller CPU demands, but have been allocated the same amount of money as longer jobs. Shorter jobs can spend more for communication resources and will submit higher bids for the communication links. This means that the shortest jobs win the auctions for communication links and migrate to the cheaper processors. An economy in which jobs use the Price preference attempts to

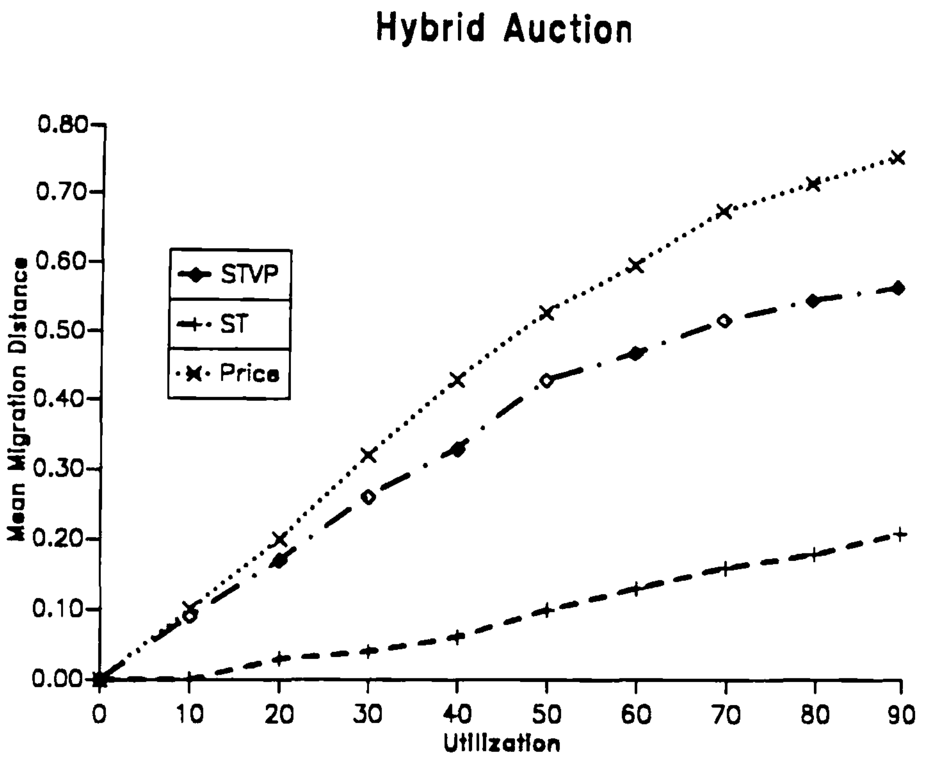


Figure 15. Job Migration Distance: This graph plots the mean distance a job migrates before receiving CPU service for each of the three preferences under the Hybrid auction model.

simulate the M/M/9 system with shortest-job-first scheduling. It moves the relatively wealthy short jobs to the cheapest (least loaded) processor.

Figure 16 on page 59 illustrates the behavior of the price preference and explains its high migration rates. There are three jobs at P_1 with CPU demands 10, 2 and 1, and each job has \$5.00. If the jobs use the price preference, all try to migrate to P_4. Assume that $S_1 = S_2 = S_3 = 0.2$. J_3 wins the auction for link e_{14} by bidding \$1.04 = \$0.10 + (0.2)(\$4.70) and this becomes the new price. Service at P_3 becomes cheapest and the remaining jobs compete for link e_{13}. J_2 wins and the price of e_{13} is set to \$1 = \$0.10 + (0.2)(\$4.50). Finally, J_1 bids \$0.50 for link e_{12} and migrates to P_2. At this point, service at processor P_1 is now the cheapest and other jobs at P_1 do not migrate. The increase in link prices make migration less preferred and prevents the system from thrashing.

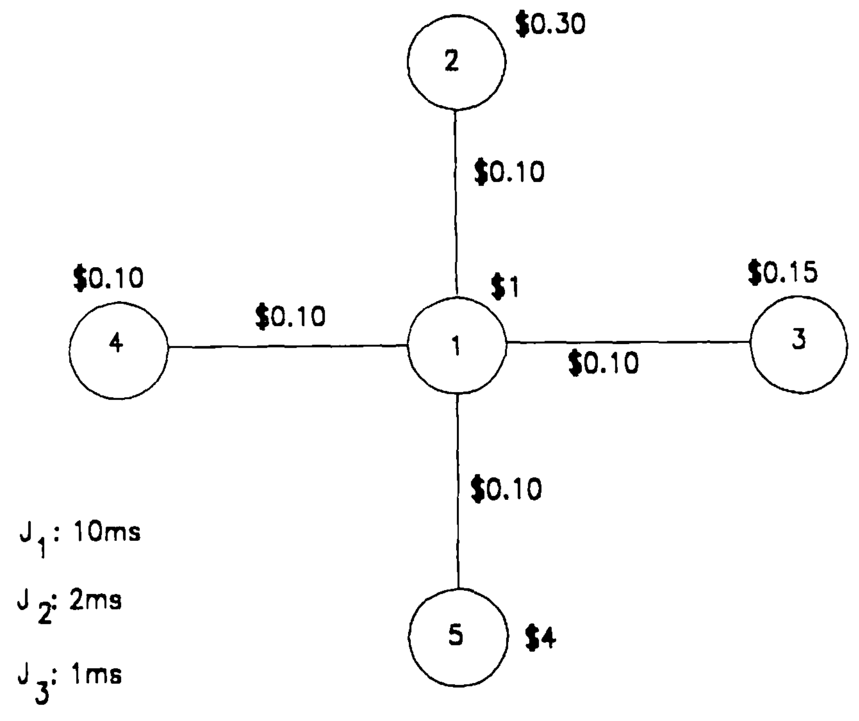


Figure 16. Choice of Migration Strategy: This figure depicts an example that explains the diversity of of migration strategies of the economy. There are three jobs at processor P_1. The prices next to resources reflect the information in the bulletin board at P_1.

When the Service Time preference is used, the behavior of the system is completely the opposite of the price preference. If all nodes have the same processing rate, local service is always preferred to migrating to another processor. This explains the low migration rates in the economies with the Service Time preference. A job wants to migrate only if the local processor is not in its budget set. These jobs are the relatively poor jobs, and since each job is allocated the same amount of money, the poor jobs are the jobs with large CPU demands.

If the jobs in Figure 16 use the Service Time preference, jobs J_2 and J_3 compete for service at node P_1 . J_3 wins with a bid of $1.80, and this becomes the new CPU price. J_2 still prefers local service and waits until the CPU becomes available. J_1 cannot afford local service and bids for link e_{14} and migrates to P_4. In this example, the Price preference generates 3 times as many migrations as the Service preference.

When the combined STVP preference is used, the economies' performance and migration rates lie between those of the Price and Service Time preferences. The behavior of this economy can be controlled by choosing the constant A that defines the STVP preference. If behavior like the Price economy is desired, A should be small to place more weight on prices. If behavior like that of Service Time is needed, A should be large. The load balancing economy spans a spectrum of possible load balancing strategies. At one extreme, the economy has a high migration rate and migrates short jobs. At the other extreme, there is a low migration rate and long jobs migrate. This provides flexibility that allows the implementer of the load balancing algorithm to choose a particular point in the spectrum by setting a single parameter.

3.3.3 Adapting to CPU vs. Communication

Figure 17 on page 61 plots the performance of the Load Balancing Economy and the HOP 1 algorithm when the speeds of the communication links have been cut in half (i.e. - 512Kb/s). The three economies use Sealed Bid auctions and the Price, Service Time and STVP preferences. In this system, the Service Time preference yields substantially better performance than either of the other preferences, and is better than the HOP 1 algorithm at high utilizations. The Service Time preference is now 25% better than the other two preferences at high utilization as opposed to 15% in the case in which the link speeds were 1Mb/s.

Figure 18 on page 62 shows the situation when the communication speeds have doubled to 2 Mb/s. Service Time and HOP1 are now the *worst*, and Price and the STVP preferences are approximately 10% better. This is in contrast to the fact that Price is the worst preference in the systems with 1Mb/s and 512Kb/s links.

The reasons for the changes in relative performance are due to the migration rates of jobs using the preferences. When the Price preference is used, the economy is very aggressive and has a very high migration rate. There is no disincentive under the Price preference for a very high migration rate when the speed of the communication links has been halved. The Price preference does not take the link and CPU speeds into consideration when ranking budget set elements. In the example of

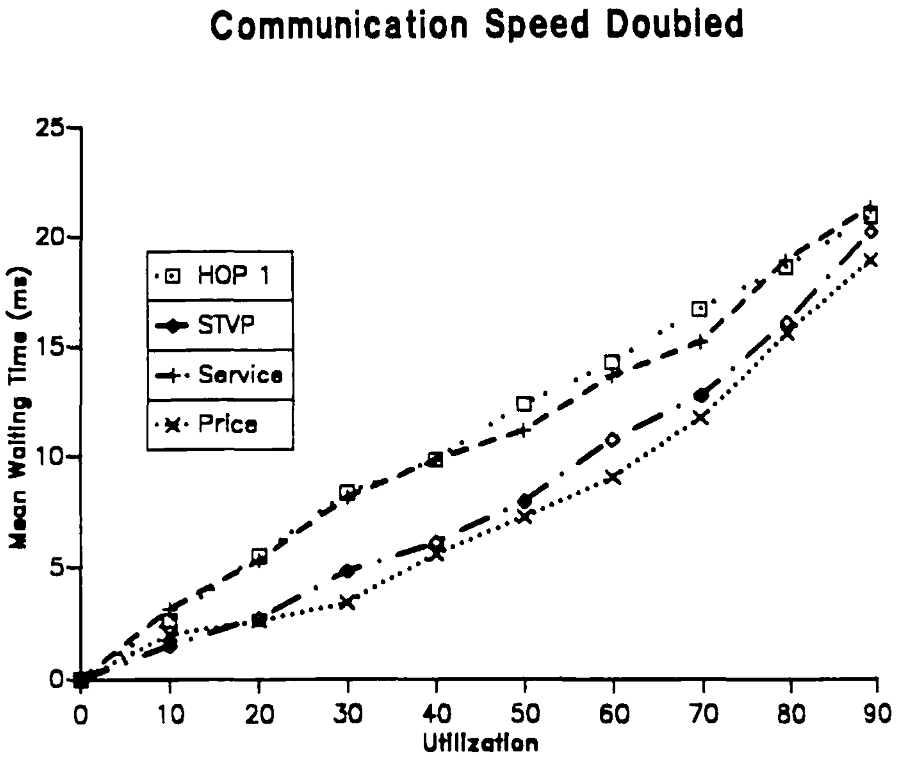


Figure 17. Communication Speed Halved: This figure plots the mean job waiting time of the three preference relations in a Sealed Bid economy and of the HOP 1 algorithm when the speed of the communication links has been halved (i.e. - 512Kb/s). Service Time and the HOP1 algorithm are the best.

Figure 16 on page 59, the sequence of job decisions under the Price preferences does not change if link speeds are decreased. The high migration rate continues to occur despite the decrease in communication speed. This strategy becomes progressively worse as link speeds decrease.

The Service Time preference generates a very low migration rate independently of the relative speed of communication to CPU power. As long as the communication speed is finite and the CPUs have the same processing rates, local service is always strictly preferred. The example of Service Time behavior in Figure 16 on page 59 is independent of link speeds. The Service Time preference does well when the communication speed has been halved since it has a low migration rate and migrates the long jobs.

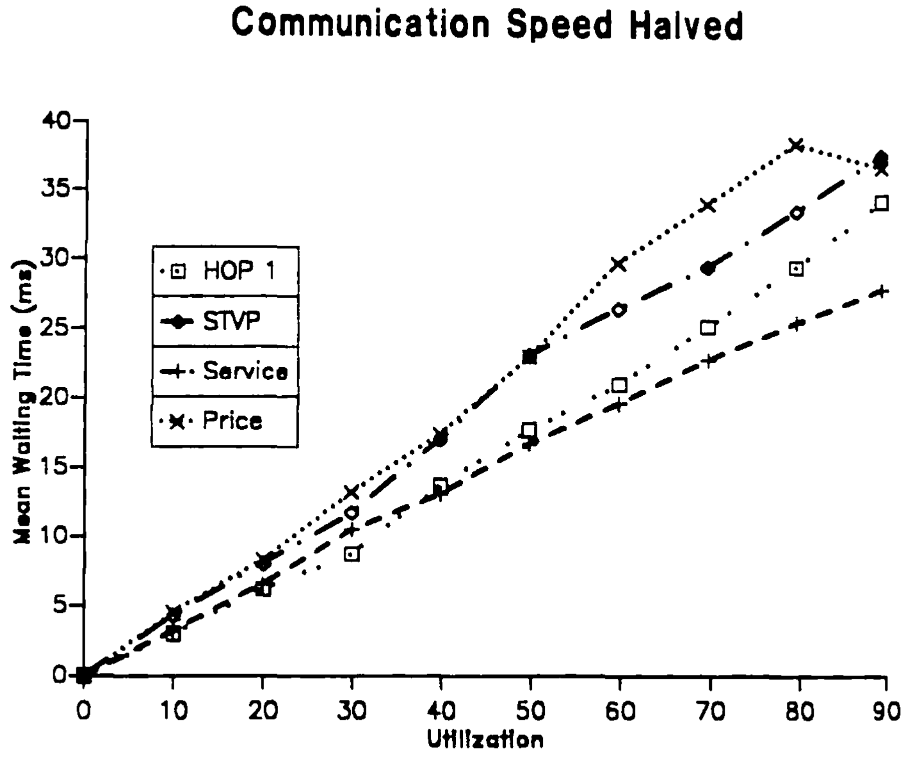


Figure 18. Communication Speed Doubled: This figure plots the mean job waiting time of the three preference relations in a Sealed Bid economy and of the HOP 1 algorithm when the speed of the communication links has been doubled (i.e. - 2Mb/s). Service Time and the HOP1 algorithm are now the worst.

When the communication speed is doubled, the aggressive migration policy of the Price preference becomes a more effective load balancing strategy. The Service Time preference cannot modify its behavior because local service is always faster than remote service.

The combined STVP preference attains performance in between the extreme cases of the Price and Service Time preferences. In the experiments presented in this section, equal weight is given to price and service, i.e. A = 1. As demonstrated in the waiting time and migration distance experiments described above, the STVP preference with A = 1 behaves like the Price economy. Although, when communication speed has been halved, the STVP preference exhibits noticeably lower waiting times than the Price preference for system utilizations in the range 60% – 80%. At 90% utilization, the Price preference cannot continue increasing its migration rate because the utilization of the communication links approaches 100% and its performance levels off.

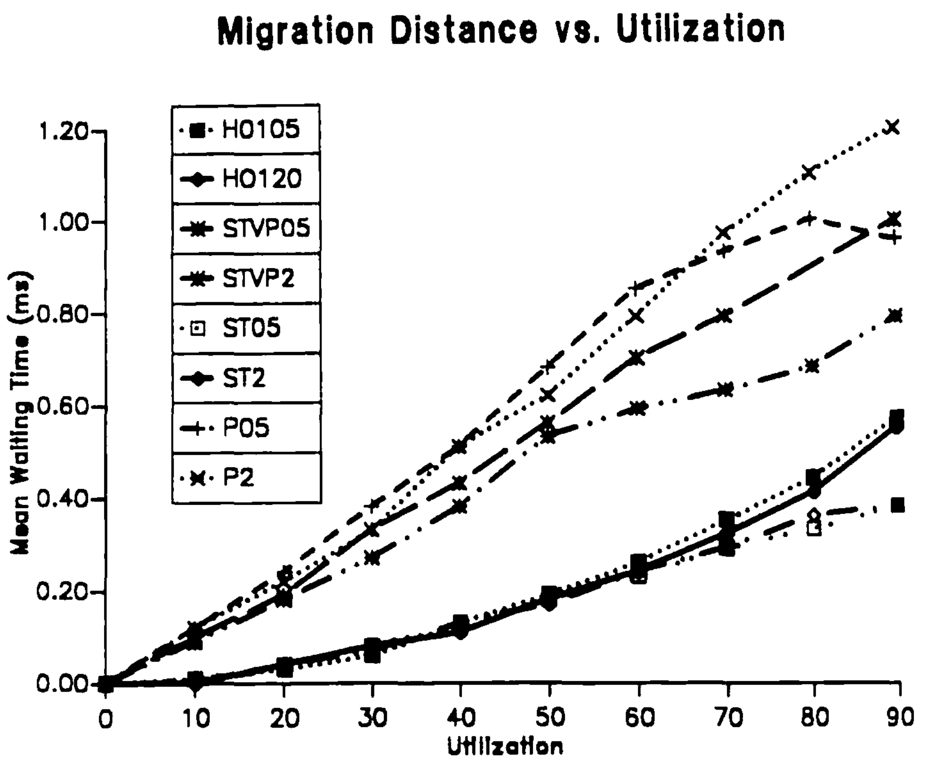


Figure 19. Migration Distances: This figure plots the mean job migration distance for three preference relations in a Sealed Bid Economy and of the HOP 1 algorithm when the speed of the communication links has been doubled (i.e. - 2Mb/s) and when the link speeds have been halved (512 Kb/s). The three preferences are represented by P = Price, ST = Service Time, STVP = STVP and HO1 represents HOP 1. A 05 following the preference symbol or HO1 represents the 512 Kb/s case and 2 represents the 2Mb/s case.

Figure 19 on page 63 shows the migration distances of the three preferences in the Sealed Bid economy and of the HOP 1 algorithm for both the cases of 512 Kb/s and 2 Mb/s communication links. In this figure P represents price, ST represents Service Time, STVP is the combined preference and HO1 is the HOP 1 algorithm. The numerical suffix 05 indicates the 512Kb/s case and 2 represents the 2Mb/s case. For example, P05 is the economy using the Price preference in a system with 512Kb/s links.

This figure shows that both the HOP 1 algorithm and Service Time economy do not adjust their migration based on the underlying ratio of communication speed to CPU rates. In both cases, the rates are nearly identical. Below 80% utilization, the Price preference does not substantially alter its migration rate either. At 80% utilization, the migration rate in the 512Kb/s system diverges from the rate in the 2Mb/s case and begins to level off. This is due to the fact that the utilization of the communication links reaches 100% for the system with 512Kb/s links, and a higher migration rate cannot be sustained.

The only algorithm that significantly alters its behavior is the STVP preference. At high utilizations, the migration rate in the 512Kb/s system is more than 20% lower than in the 2Mb/s system. Despite the fact that the weighting factor A does not change, the economy does alter its behavior. Changing the weighting factor can be used to tune the economy to optimize its performance for a given ratio of communication to CPU power.

The example of Figure 16 on page 59 can demonstrate the effects of the parameter A on the economy's behavior. Assume that the migration delay is 2ms for all jobs and links. Job J_3 attempts to migrate at any choice of A because P_1 is not in its budget set. For J_2 to prefer service at P_4, it must be the case that $6A + 0.40 < 2A + 2$, which means $A < 0.40$. At this value, $5A + 0.30 > A + 1$ and J_3 still prefers local service. For J_3 to prefer migration, we must have $5A + 0.30 < A + 1$, or $A < 0.175$. At this value, $6A + 0.50 < 2A + 2$ and J_2 migrates to P_3. The choice of A determines if 1, 2 or 3 migrations occur in this example.

In summary, the Load Balancing Economy is very versatile and offers a spectrum of load balancing strategies. Different strategies are better for differing ratios of communication to CPU speeds. It is possible to tune the economy to this ratio by adjusting a single weighting factor, and this can offer substantial performance improvements over a non-economic load balancing algorithm. The main disadvantage to this approach is that manual intervention is required to adjust the weighting factor.

3.3.4 Flow Control

The Load Balancing Economy implements flow control at high utilizations. Figure 20 on page 65 plots the job throughput as a function of system utilization. This figure shows the three preference relations in a Hybrid auction economy, and the HOP 1 algorithm. Above 60% utilization, the economies' throughputs taper off. This is caused by the intense competition at high utilizations. Relatively poor jobs or jobs that make poor resource purchasing decisions enter a situation in which they cannot acquire the resources needed to complete processing and leave the system. This

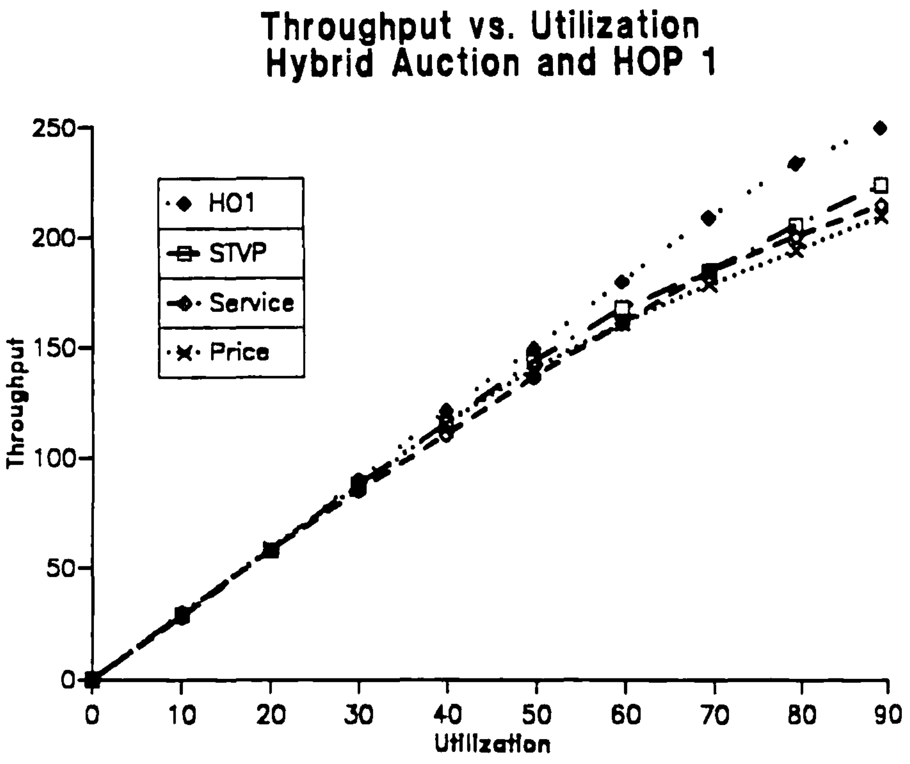


Figure 20. Flow Control: This figure plots the job throughput as a function of system utilization. The figure plots the throughput of the Hybrid Auction economies with all three job preferences, and the throughput of the HOP 1 algorithm.

improves the performance of the wealthier jobs by decreasing the effective utilization of the system. Throughput is decreased to improve response time.

The form of flow control implemented has some drawbacks. First, the flow control does not *prevent* jobs from entering the system. Instead, the jobs enter the system and are denied resources. A user that submits a job does not know if the job will be serviced or not because there is no way to tell if the job will eventually receive service. Secondly, the jobs that are flow controlled remain in the system but cannot purchase any resources. The amount of storage used to hold idle jobs increases monotonically over time. Obviously, this cannot be allowed to occur indefinitely. Potential solutions to these problems are discussed in an appendix. The main problem with the flow control mechanism implemented by the load balancing economy is that it cannot easily be controlled.

Unlike adapting to the CPU/Communication speed ratio, there is no effective way to set the desired throughput/response time goal. Correcting these deficiencies is an area for future work.

3.3.5 *Wealth, Priority and Learning*

A Job J's bidding rule has two parameters. The first is S_J, which determines J's use of surplus funds. The second parameters is a feedback factor δ_J. Each time J wins an auction, it sets $S_J = S_J - \delta_J$. When J loses, it updates $S_J = S_J + \delta_J$. The use of the feedback δ_J implements a simple learning mechanism. The goal is to learn the minimum use of surplus funds needed to win an auction. When resource demands are low, J wins auction and decreases S_J. During periods of high demand, J loses and increases S_J.

One of the goals of applying economic concepts to computer science resource management problems is to be able to define an agent's priority by the amount of wealth allocated to it. Ideally, the wealthier jobs should experience lower response times than the poor jobs. Figure 21 on page 67 shows the results of an experiment testing the effectiveness of wealth as priority. In this test, a job entering the system is assigned a random amount of money independently of its CPU demand. In this figure, the X-axis represents a job's ***relative wealth***. If job J has initial wealth m_J and CPU demand μ_J , its relative wealth is $\frac{m_J}{\mu_J}$. The Y-axis in Figure 21 on page 67 represents mean job waiting time. The two curves plotted are both for the Hybrid Auction-Price economy. In the upper curve, the feedback parameter δ_J in each job J's bidding strategy is 0, i.e.- no learning occurs. The second curve plots the performance when $\delta_J = 0.05$.

When $\delta = 0$, relative wealth is a very effective priority scheme. Mean job waiting time is a monotonically decreasing function of relative wealth in the interval [0, 0.80] . Above 0.08 the mean job waiting time levels off, and there is no distinction between the wealthy and the supper wealthy.

The lower curve in Figure 21 on page 67 depicts the case of $\delta_J = 0.05$ (learning occurs). In this case, wealth is completely ineffective as a priority scheme. However, all jobs are substantially better off than when $\delta_J = 0$. The reason for the ineffectiveness of wealth as a priority is a result of the learning process. A job J that has spent time in the system has lost several auctions and increases S_J . It more aggressively uses its surplus funds and submits bids higher than the asking price set by the

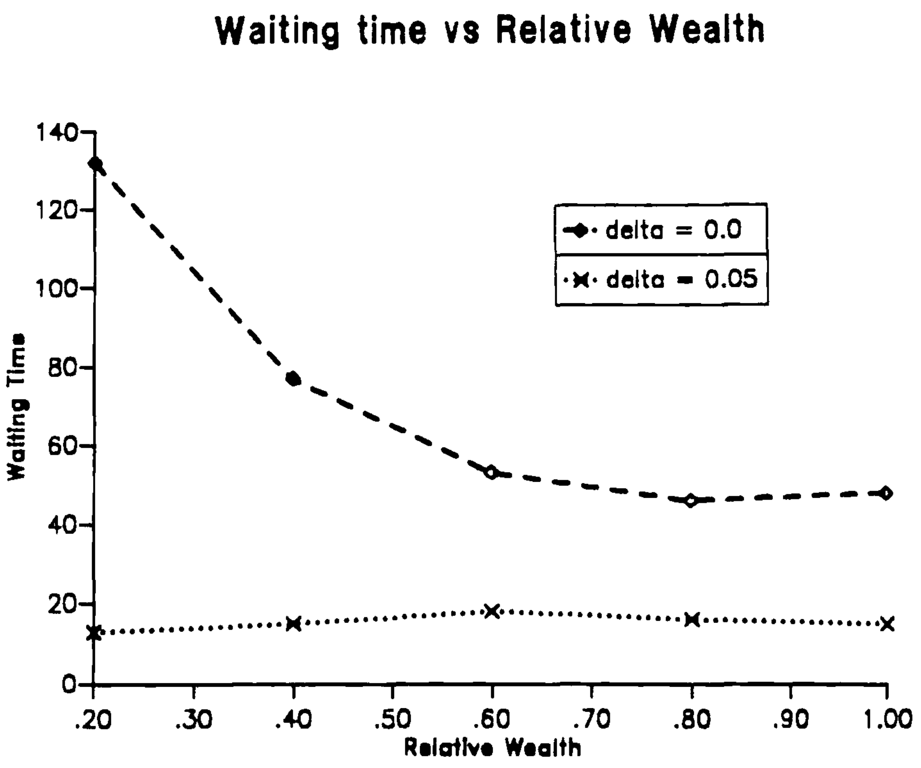


Figure 21. Wealth as Priority: This figure plots the mean job waiting time as a function of relative wealth. Relative wealth is the job's initial allocation of money divided by its CPU demand. The two curves in the figure represent cases in which there is no feedback from auctions ($\delta = 0$) and there is feedback from auctions ($\delta = 0.05$)

processors. Younger jobs have lost fewer auctions and use less of their surplus funds. So, the younger jobs lose to the older jobs.

When $\delta = 0$, the price for which a resource sells in an auction is not a good indicator of the demand for the resource. In a Sealed Bid auction, if the bidding constant S_j is the same for all jobs, the price for a resource is set by the wealthiest job. The number of other jobs demanding the resource and their individual allocations of money have no effect on the price. The same problem occurs in a Hybrid auction. The only difference is that the price is set by the *two* wealthiest jobs. The number and wealth of the other jobs demanding the resource is not reflected by the price. Figure 22 on page 69 depicts these situations for the Sealed Bid and Hybrid auction models. This explains the poor performance of the Load Balancing Economy when $\delta_j = 0$. The prices charged for the resources do not reflect the demand for them, and the CPU price is not a good measure of a processor's load.

Setting $\delta > 0$ partially corrects this deficiency of the auction model. In this case, the price of a resource is determined by the age of the job purchasing it. When resource demands are high, queueing delays increase and the ages of jobs purchasing resources increase also. As a job J loses auctions it increases S_J, which in turn increases the sale price of the resource. Price becomes a better indicator of a processor's load. This explains the improved performance of the economy as a whole when $\delta = 0.05$.

Setting $\delta_J > 0$ does not completely solve the problem of price being a poor indicator of load. For example, a surge in load does cause a price increase, but this increase peaks *after* the surge has passed and as the load starts to decrease. Figure 23 on page 70 depicts an an example of this problem that can occur even if $\delta_J > 0$. In part *a*, the system is empty and $P = 0$. In part *b*, 3 jobs arrive simultaneously and each job has \$10, $S_J = 0.25$ and $\delta_J = 0.05$. After the first sale $P = \$2.50$. The losing jobs J_2 and J_3 increase S_J by 0.05. In the next round (part *c*), the resource sells for $\$4.75 (= \$2.50 + 0.3 \cdot (\$7.50))$. If no more jobs arrive, when job J_3 purchases the resource (part *d*) the price becomes \$6.59 but the load is only $\frac{1}{3}$ of the load when the resource was selling for \$2.50.

Wealth as a priority suffers from the same problem as flow control of the preceding section. Very sophisticated behavior is exhibited and this behavior is more sophisticated than traditional load balancing algorithms. However, the behavior is not controllable.

3.4 Comparison with Related Work

This section compares the load balancing economy with previous load balancing algorithms. We focus on the novel contributions of the load balancing economy. For more detailed surveys on algorithms, readers are referred to [29, 74]. Actual implementations of distributed systems that support load balancing are discussed in [6, 46, 47].

The performance results presented in this chapter and previous work [26, 27, 44, 54, 64, 65] demonstrate that the interprocessor communication medium has a major impact on the effectiveness of load balancing. The communication medium has two sources of delay. The first is the static

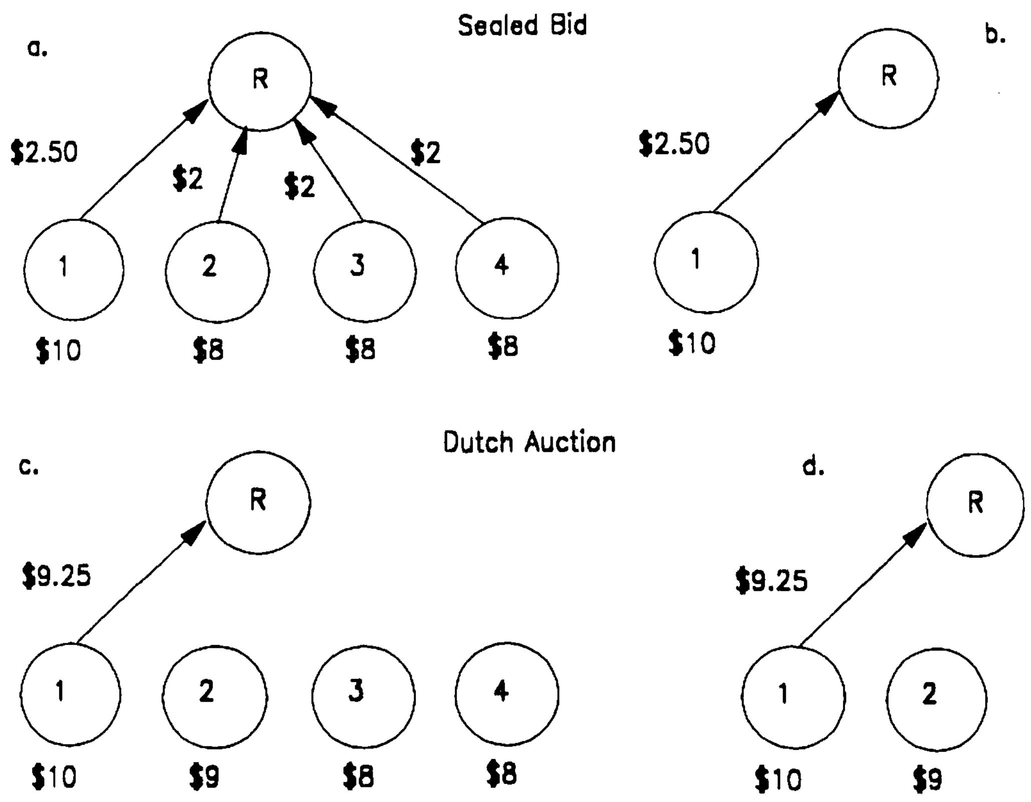


Figure 22. Auctions and Prices: This figure shows why auction prices do not accurately reflect demand. In this figure, 4 jobs bid for a resource R and all have $S = 0.25$. Values below jobs represent wealth and values next to arrows represent bids. In the Sealed Bid auction, the sale price is $2.50 whether or not J_2, J_3 and J_4 demand the resource. In the Hybrid auction, J_1 and J_2 compete driving the price to $9.00, and J_1 wins with a bid of $9.25. Jobs J_3 and 4_3 have no effect on the price.

delay of executing the communication protocols at the sender and receiver, and the transmission delay on the link. These are the factors included in the definition of ST_k in the job preference relations. The second source of delay is queueing delay at these resources. Some previous work has ignored the effects of the communication delay on performance [34, 35, 53, 74, 91]. Two constructive approaches to dealing with communication delays have studied. The first is to parameterize the algorithm to alter its behavior based on communication costs [26, 27, 36]. The second is the use of different algorithms for different configurations [54, 64, 65].

The load balancing economy deals with the two sources of communication delay with different approaches. The static delay of job migration relative to CPU processing is solved by setting the

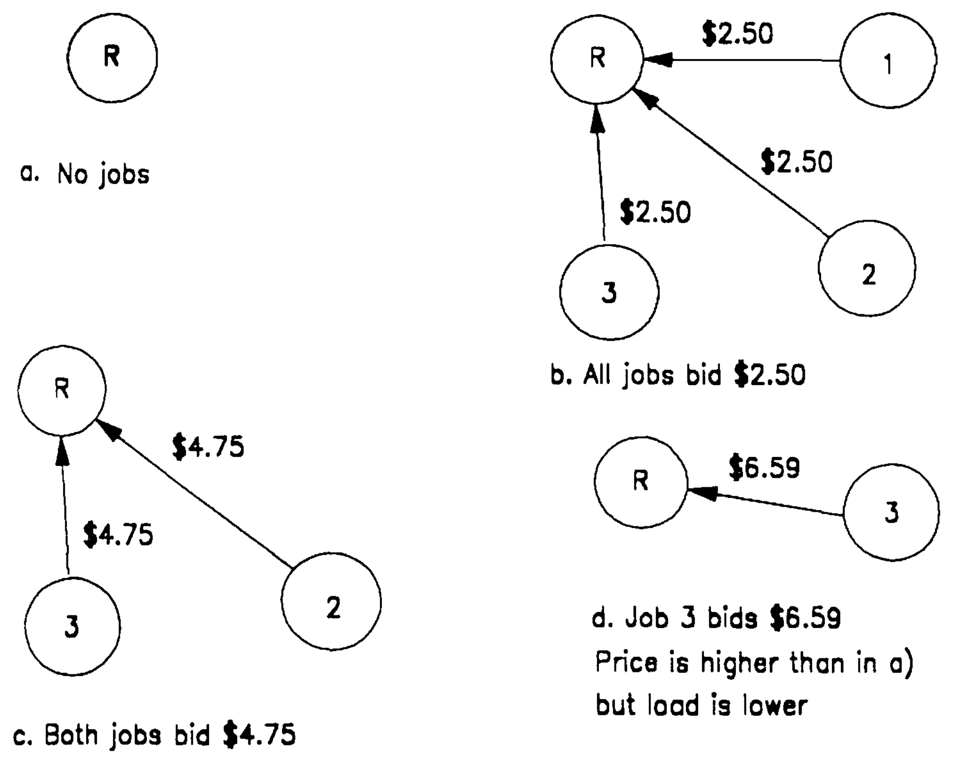


Figure 23. Price Pathology: This figure shows how the price for a resource can be a poor indicator of the demand for the resource. In this figure, each job has \$10, $S_j = 0.25$ and $\delta_j = 0.05$. In the first auction, the price is \$2.50. By the third auction, when the surge has passed and the load is lower, the price is \$6.59.

parameter A in the STVP preference. The performance experiments in this chapter demonstrate that vastly different load balancing strategies can be implemented by different values of A. Parameters in previous algorithms are typically integers, while A can be any real number. This should allow finer control, but this has not been addressed in this dissertation.

Delays caused by queueing at communication resources is handled by the price system. Higher demand for communication resources increases the link prices. This decreases the preference for job migration. The experiments with the hybrid auction model demonstrate that the load balancing economy decreases migration rates as the competition for communication resources increases. This process does not require external control to set parameters. The load balancing economy is self-regulating, and this behavior is not present in previous algorithms.

Stability has been identified as a problem for dynamic load balancing algorithms [12, 86]. An algorithm is unstable if it can enter a state in which jobs are being migrated to balance loads but no productive processing is being done. The phenomenon is similar to thrashing in virtual memory [23]. Two approaches have been taken to ensure stable behavior. The first uses global information to periodically compute assignments of work to processors to minimize a system wide performance objective [34, 35]. The resulting assignment is fixed during the next interval and instability is avoided. The overhead of acquiring consistent global information can make this approach unworkable. The second solution to the stability problem is augmenting the load balancing algorithm with a heuristic that prevents unstable states [26, 27, 36, 53, 54, 64, 65]. Two common heuristics are:

1. Each job J can only be migrated M times by the algorithm. This prevents the system from infinitely migrating jobs.

2. Let L_i, L_j be the loads of processors P_i, P_j . A job can be transferred from P_i to P_j only if $|L_i - L_j| > T$, where T is a threshold parameter greater than the load of any single job. This disallows migrations that would reverse the load imbalance, and prevents P_i and P_j from entering a state in which they infinitely migrate jobs between each other.

There are two disadvantages to the use of heuristics. The first is increased complexity. The second disadvantage is that the heuristics are pessimistic. The heuristics disallow migrations that may improve performance to avoid the possibility of instability.

The load balancing economy is inherently stable. A job pays a non zero fee each time it crosses a link. So, infinite migration is not possible. This mechanism implements a more general version of heuristic 1 above. The link prices increase when there is competition. This has the desirable advantage that the distance a job is allowed to migrate decreases as the demand for communication resources increases. The experiments with the hybrid auction model demonstrated that increased competition for communication resources decreases the preferability of migration. This also prevents instability. The sealed bid auction model is less effective in this respect.

Load balancing algorithms can be partitioned into to classes: *sender initiated* and *receiver initiated* [27, 74]. In a sender initiated algorithm, over loaded processors look for under loaded processors to which work can be sent. In a receiver initiated algorithm, under loaded processors search for over loaded processors from which work can be obtained. It has been shown that at high average system load receiver initiated algorithms achieve better performance, and at low average load sender initiated are better [27, 74].

The load balancing economy is both sender and receiver initiated and can encompass the benefits of both models. A heavily loaded processor charges a higher CPU price which causes jobs to look for cheaper, less loaded processors. This implements sender initiated load balancing. The less loaded processors advertise their lower CPU prices to attract migrating jobs (receiver initiated). The HOP 1 algorithm is sender initiated. A comparison between the load balancing economy and a receiver initiated algorithm is an area of future work.

The load balancing economy limits complexity in several ways. First, all decision making is decentralized and this decentralization removes complexity due to cooperative protocols. The only interaction between processors is inserting advertisements into bulletin boards. The processors do not make decisions based on this information. All local auctions and price updates are performed independently from other processors. The decision making rules of the jobs are independent.

The decentralized decision making allows a very modular implementation. It is possible to change a processor's auction model without effecting the policies of any other processor. In our experiments, switching between sealed bid and the hybrid auction models did not require any changes in the job's rules. Similarly, varying a jobs preference relation does not require any modification to other jobs or the processors. This modularity facilitates transparent addition of processor and job rules and software maintenance.

Finally, the load balancing economy decreases complexity be eliminating the need for special heuristics or alternate algorithms. Tuning the algorithm for the ratio of CPU vs Communication, preventing instability and implementing sender/receiver initiated load balancing are inherent properties of the economy.

3.5 Summary

This chapter presented the load balancing economy. A simulation study showed that this economy achieves substantially improved system performance compared to a distributed computer system in which load balancing is not implemented. The economy was also compared with a non-economic load balancing algorithm. It was shown that performance improvements achieved by all variations of the load balancing economy are competitive with the improvements of the non-economic algorithm, and in some cases the load balancing economy is strictly better.

The load balancing economy was shown to implement a very broad range of load balancing strategies. The mean distance a job migrates in search of CPU service changes by a factor of 3 for different versions of the economy. The decision as to which type of job (short vs. long) should migrate to balance loads also changes. This adaptability allows the economy to be optimized for the ratio of communication to CPU power in the underlying distributed system.

The load balancing economy implements a form of flow control at high utilizations. Job throughput is throttled to improve mean job response time. We studied the effectiveness of a the amount of money allocated to a job as a definition of priority. The role of learning and its interactions with the use of wealth as a priority scheme were addressed also.

We discussed how the load balancing economy limits complexity compared to previous load balancing algorithms by eliminating the need for multiple heuristics. Previous algorithms deal with specific sub-problems of load balancing by applying ad hoc heuristics. These heuristics are not necessary in the load balancing economy. The decentralized decision making in the load balancing economy also limits complexity by implementing a set of modular, disjoint algorithms.

We discussed three shortcomings of the load balancing economy. The first is the potentially high overhead of auctioning idle resources. The second is the inability of auctions to set resource prices that accurately reflect the demand for the resources being sold. These two problems motivate our use of different pricing mechanisms in the later chapters.

The third shortcoming is the assumption that a job's CPU service demand is known when the job is submitted. In general, economic models assume that individual agents know their resource demands. This may not always be true for the load balancing problem, but there are environments in which this assumption is realistic.

3.6 Appendix

3.6.1 Issues

The discussion of the design of the load balancing economy and of the performance experiments pointed out disadvantages of the economy and raised other issues. These issues are addressed in this appendix. The solutions to the disadvantages and issues raised are left for future work.

3.6.1.1 Knowledge of Resource Demands

The jobs' behavioral rules rely heavily of the assumption that each job J knows its resource demands (μ_J, $ReqBC_J$, $ReqBC_J$). The assumption that individual agents know their resource demands is an integral part of microeconomic models [40]. Modifying the algorithms to deal with incomplete information is left as an area of future research.

Modifying the economy to deal with incomplete information faces two problems which are orthogonal to the economic algorithms. If no information about a job J's CPU demand is known, the information must be inferred from resource consumption as the job resides in the system. For example, jobs can purchase time slices. The job's total CPU demand can be estimated based on the number of slices already purchased. This approach is similar to multilevel feedback queueing for CPU scheduling in operating systems [23]. Bryant and Finkel [12] studied a load balancing model in which job CPU demands were estimated based on service received. The main problem with this approach is that load balancing must be *preemptive*, and migrate actively running jobs. It is typically assumed that the communication and CPU overhead of preemptive load balancing is prohibitive [75].

An alternate approach is to estimate the resource demands based on previous execution, or based on job classes. This limits the generality of the system. Furthermore, computing these estimates is time consuming and requires substantial resources itself [30].

3.6.1.2 Auctions

Section 3.5 pointed out several disadvantages of using auctions to allocate resources. An additional problem is overhead. Each time a resource is allocated all locally residing jobs participate in an auction to determine which job gets the resource. The overhead is a greater problem under the hybrid model because there are multiple rounds of bidding. This overhead and the problems discussed in section 3.5 have led us to study alternate models in chapters 4 and 5. The data management economy does successfully use an English Auction, however.

3.6.1.3 Advertisements

There is potentially an overhead problem associated with the processors' advertisement policy. Processing node P_i broadcasts every CPU price change to all neighbors. The advertisement messages are very small, and in a point-to-point network they can be piggybacked on normal data transfer messages. The processing overhead for receiving an advertisement and updating the bulletin board is also quite low. So, the advertisement policy's overhead is tolerable in point-to-point networks.

Livny [64, 65] has shown that broadcasting all state changes can have a prohibitive effect on the performance of distributed systems with a broadcast interconnect medium. In such an environment, the load balancing economy requires heuristic rules to limit the amount of advertisements sent. For example, only price changes greater than a threshold Δ are sent. These heuristics add complexity, however.

3.6.1.4 Price Guarantees

When job J decides to migrate from processor P_i to P_j it bases its decision on the CPU cost of P_j contained in P_i's bulletin board. This price is not guaranteed, and can change while J is migrating. Furthermore, J participates in an auction for the CPU after it arrives at P_j and may not pay the price advertised at P_i. These factors can generate fruitless migrations that consume communication resources an incur overhead.

An alternate approach is to guarantee prices. In this case, if J sees price c_j advertised in P_i's bulletin board, P_j must honor this commitment when J arrives. This policy greatly complicates the economy, however. Since prices are set by auctions, resource prices change rapidly. If n jobs arrive at P_j, it is possible that each job has seen a different advertised price. Processor P_j must decide which job to service next. Furthermore, it is not clear what price P_j should advertise when allocating the CPU to one of these jobs. Finally, if a job was guaranteed a price c_j' which is greater than the current CPU price c_j, should J be allowed to spend less? We avoided implementing guaranteed pricing in the load balancing economy due to the complexity it incurs.

3.6.1.5 Flow Control

Section 3.4 pointed out two shortcomings of the flow control policy implemented by the load balancing economy. These are:

1. Flow control does not prevent entry into the system. Jobs enter the system, but never execute.
2. The buffer space required to hold these jobs increases monotonically because jobs are never terminated.

There are two potential solutions to these problems. The first is guaranteeing prices, which is discussed above. A second approach is to charge queued jobs rental fees for their buffer space. This would mean that every job submitted either receives service and completes, or is aborted when its funds are exhausted.

3.6.1.6 Returning Jobs

The description of the job behavior in section 2.1 did not describe how jobs return to the processor at which they entered the system. The algorithm for implementing this requirement is described here.

Each time a job crosses a link seeking CPU time, it sets aside ("banks") money to buy the same link on the way home. Job J may also have some surplus funds after its CPU time has been purchased. This surplus does not include the money "banked" to buy the communication bandwidth for the journey home. In the example of Figure 9 on page 49, assume that when J_1 gets to P_2, the CPU price has fallen to \$0.01 per millisecond. J_1 allocated \$3.00 to buy the CPU, but spends only \$0.30; it has a surplus of \$2.70. J_1 also set aside \$1.00 to buy the link from P_2 to P_1 on the way home. J_1 actually has \$3.70 with which it can buy link e_{21}.

If the link price is equal to or lower than the initial estimate, the job will only use the money set aside to try to buy the link. Otherwise, it will use a percentage of its surplus based on the value of S_J. Returning to our example, if P_2 is charging \$0.0007 per byte to get to P_1, J_1 will have a surplus of

$$\$1.00 - 0.0007 \bullet 1000 = \$0.30$$

and will bid

$$1000 \bullet 0.0007 + (\$0.30) \bullet S_2.$$

If, however, the price is \$0.002, J_1 cannot afford the link with the \$1.00 set aside. So, it will bid

$$1000 \bullet 0.002 + (\$3.70 - \$2.00) \bullet S_2.$$

This example illustrates the important aspects of the rules jobs use when returning to their origin. These rules are quite simple, and should be sufficient if jobs rarely migrate more than one or two hops. Our simulation studies show that the mean job migration distance is usually less than 1 hop.

3.6.2 The Hop 1 Algorithm

This section presents the modified HOP 1 algorithm [64]. In its basic version, the HOP 1 algorithm does not use the service times of the individual jobs when making load balancing decisions. Each processor P_i maintains a list of the jobs currently queued locally waiting for CPU service. This list is denoted $N(i)$. Since the performance goal is to minimize average response time, and the service times of the jobs are known, the optimal scheduling policy is *Shortest Job First* [51]. To implement this policy, the list $N(i)$ is sorted from smallest μ_J to largest μ_J. When P_i's CPU becomes idle due to a job completion, the first job in $N(i)$ is assigned the processor.

If processor P_k decides to send job J to processor P_i, it first sends a *Reservation Message* $<J, \mu_J>$ to P_i, and then initiates the migration. The reservation message arrives much more quickly than the migrating job and allows P_i to make load balancing decisions based on locally queued jobs, and on jobs that will soon arrive. $R(i)$ is the set of reservation messages that P_i has received from all neighbors. When job J arrives, $<J, \mu_J>$ is deleted from $R(i)$ and inserted in $N(i)$.

The load L_i of processors P_i is defined as

$$L_i = \sum_{J \in N(i)} \mu_J + \sum_{J \in R(i)} \mu_J.$$

Each change in L_i is broadcast to all of P_i's neighbors. For each neighbor P_k, P_i maintains the last reported load from P_k in $L_i[k]$.

Processor P_i attempts migrations if:

1. There is at least one neighbor P_k with $L_i[k] < L_i$, and
2. $\#(N(i)) > T$, i.e. - more than T jobs are queued locally. T is the *Transfer Threshold*, and is a parameter of the algorithm.

P_i migrates jobs to the neighbor P_k with minimal $L_i[k]$.

We implemented two policies to select the job to migrate. They are:

1. **First Fit Selection Policy**: Select the job with the smallest CPU time in $N(i)$, denoted M, provided that

$$L_i[k] + \mu_M \leq L_i - \mu_M.$$

   In other words, do not pick any job if doing so would reverse the load imbalance.

2. **Best Fit Selection Policy**: Choose the *largest* job (denoted B) that satisfies the condition

$$L_i[k] + \mu_B \leq L_i - \mu_B.$$

If the P_i picks job J for migration to P_k, the following steps are performed:

1. Send Reservation Message $<J, \mu_J>$ to P_k.
2. Initiate the migration of job J to P_k.
3. Delete $<J, \mu_J>$ from $N(i)$.
4. Set $L_i = L_i - \mu_J$
5. Set $L_i[k] = L_i[k] + \mu_J$

The HOP 1 algorithm is invoked at P_i when one of the following events occurs: 1) A local job arrival, 2) A Reservation Message arrival, 3) A job completion, or 4) A migrating job arrival. The HOP 1 algorithm executes repeatedly until no further migrations are possible. This means that either $\#(N(i)) < T$ or P_i is less loaded than its neighbors. The algorithm also stops if no job can be migrated without reversing the load imbalance.

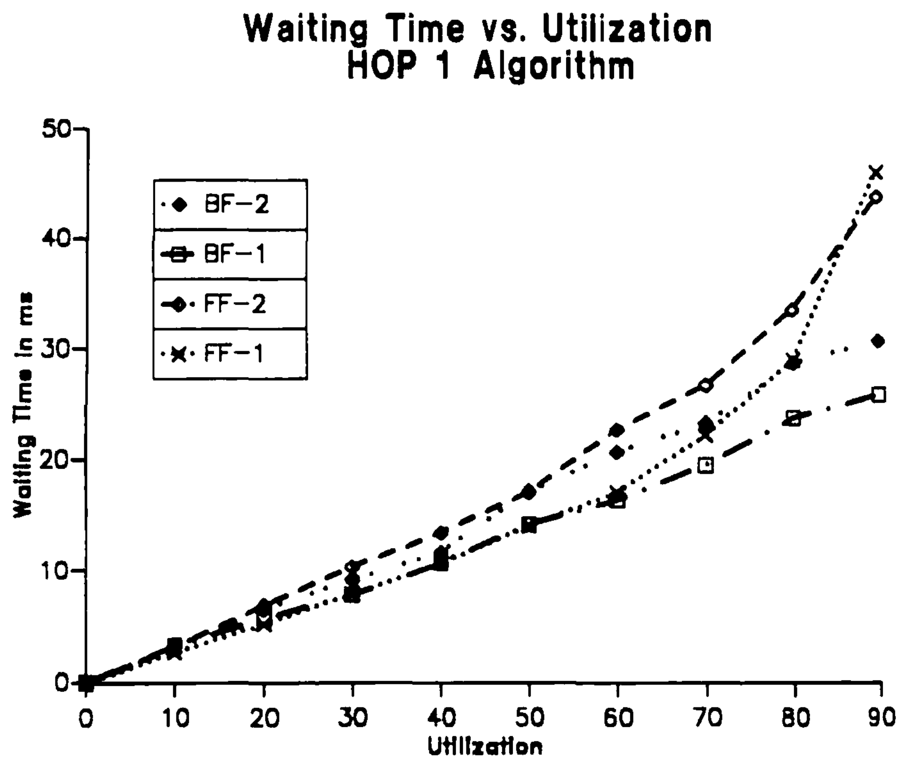


Figure 24. Modified HOP 1 Algorithm: This figure plots the performance of the HOP 1 algorithm for possible choices of the Selection Policy (First Fit (FF) and Best Fit (BF)), and for the threshold T in the Transfer Policy. The thresholds depicted are $T = 1$, $T = 2$. The first two letter in the legend indicate the Selection Policy, and the number following represents T.

The modified HOP 1 algorithm was simulated in the same distributed system as the load balancing economy. Figure 24 on page 79 plots the performance of the modified HOP 1 for the following choice of parameters:

1. Best Fit, T = 1 (BF-1)
2. Best Fit, T = 2 (BF-2)
3. First Fit, T = 1 (FF-1)
4. First Fit, T = 2 (FF-2)

The best choice is Best Fit with transfer threshold $T = 1$. This is the choice of values that was used in the previous performance comparisons with the load balancing economy. The performance of the load balancing economy for all choices of preferences and auction models is competitive with (or better than) the *best* choice of parameters for the modified HOP 1 algorithm. Other choices of parameters for HOP 1 yield performance worse than all 6 combinations of preference relations and auction models for the load balancing economy.

4.0 The Flow Control Economy

This chapter presents an economy that implements flow control in a virtual circuit based computer network. The agents in this economy are the virtual circuits (VCs) competing for communication resources and the link controllers that set the prices for these resources. We present decentralized, synchronous behavior algorithms for the agents in our economy. The allocation of resources to VC agents computed by these algorithms is proven to be *Pareto-optimal* [40]. This is a new definition of optimality for the flow control problem and we discuss the benefits of this definition. One of the goals of flow control is to fairly allocate communication resources to VCs [9]. Pareto-optimality provides a new definition for fair allocations of link capacity to virtual circuits. We present a formalization for fairness definitions, and we compare previous definitions to Pareto-optimality. We prove the existence of a Pareto-optimal allocation for arbitrary networks. Finally, the results of a simulation study of the convergence behavior of the economy are presented. This study demonstrates that the economy rapidly converges to a Pareto-optimal allocation over a broad range of parameters for realistically large networks.

This chapter is structured as follows. Section 1 presents the network model used. This section also describes the economy in detail. Section 2 proves that the flow control economy computes Pareto-optimal allocations of link capacity to VCs. Additionally, this section presents a formalization for defining fair resource allocations and compares the economy to previous work with respect to fairness. Section 3 uses concepts from mathematical economics to prove the existence of a feasible, Pareto-optimal allocation for arbitrary flow control economies. In section 4, the results of an extensive simulation study measuring the economy's convergence to optimal allocations are presented. Section 5 compares the economy to previous work on flow control, and section 6 is a conclusion. Several lemmas used in the proofs of this chapter's theorems are proved in an appendix.

4.1 Network and Economy

In this section, we define the network model and the economy. Model the network as a graph $G = \langle V, E \rangle$, where $V = \{1, 2, \ldots, K\}$ is a set of nodes and $E = \{1, 2, \ldots, M\}$ is the set of edges representing the network links. We associate two numbers with each edge $i \in E$: A *capacity* C_i (measured in data units per second), and a *link supply* S_i such that $S_i < C_i$. The link supply sets an upper bound on the link utilization to assure that the queuing delays are bounded. This does not limit generality because S_i arbitrarily close to C_i is allowed. The goal of the flow control economy is to allocate the link supplies among virtual circuits (VCs).

Let $A = \{1, 2, \dots, N\}$ denote the set of VCs. For each VC $a \in A$ there is an associated path of edges in G,

$$P_a = l_{a,1}, l_{a,2}, \dots, l_{a,m(a)}, \text{ with } l_{a,i} \in E.$$

VCs play the roles of *consumer agents* in the economy and seek allocations of link capacities. An *allocation* of capacity is a vector $\vec{x} = \langle x_1, x_2, \dots, x_M \rangle$, such that $x_i \geq 0$. Each link $i \in E$ has a *supplier agent* (network node) that charges VCs using i. A *price system* is a vector $\vec{p} = \langle p_1, p_2, \dots, p_M \rangle$, where $p_i \geq \varepsilon > 0$. We restrict prices to be strictly positive in order to avoid pathological behaviors. The assumption that $p_i > 0$ is used in later sections. This assumption does not limit generality because p_i arbitrarily small is permitted.

Each VC a has a binary *preference relation* [40] denoted $\succcurlyeq_a$ over the allocations; $\vec{x} \succcurlyeq_a \vec{y}$ denotes that VC a prefers allocation $\vec{x}$ at least as much as allocation $\vec{y}$. If $\vec{x} \succcurlyeq_a \vec{y}$ and $\vec{y} \succcurlyeq_a \vec{x}$, VC a is *indifferent* between $\vec{x}$ and $\vec{y}$ (denoted $\vec{x} \sim_a \vec{y}$). If $\vec{x} \succcurlyeq_a \vec{y}$ but not $\vec{x} \sim_a \vec{y}$, then VC a strictly prefers $\vec{x}$ ($\vec{x} \succ_a \vec{y}$).

We define $\succcurlyeq_a$ to reflect two features of an allocation $\vec{x}$: Its throughput $T(\vec{x})$ and its worst case average delay $D(\vec{x})$. A VC a attempts to maximize $T(\vec{x})$ until it reaches its individual throughput goal γ_a chosen by the VC's user, and then tries to minimize $D(\vec{x})$. The relation $\succcurlyeq_a$ is defined to capture this goal as follows:

1. If $T(\vec{x}) \leq \gamma_a$ and $T(\vec{y}) \leq \gamma_a$, then if $T(\vec{y}) \geq T(\vec{x})$, $\vec{y} \succcurlyeq_a \vec{x}$.
2. If $T(\vec{x}) < \gamma_a$ and $T(\vec{y}) \geq \gamma_a$, then $\vec{y} \succ_a \vec{x}$.
3. If $T(\vec{x}) \geq \gamma_a$ and $T(\vec{y}) \geq \gamma_a$, then if $D(\vec{x}) \geq D(\vec{y})$, $\vec{y} \succcurlyeq_a \vec{x}$.

This preference relation can be tailored to model the diversity of throughput-delay goals in a computer network supporting many types of VCs. A circuit supporting a throughput oriented application (e.g. - file transfer) is modeled by setting $\gamma_a = \infty$. The preference relation also models applications with low throughput demands and tight delay constraints such as a VC connecting a workstation to a database. In this case, the throughput goal γ_a is the mean transaction request/response packet size divided by the user's "think time." A VC carrying voice can set γ_a to the value needed to carry the voice signal (64 Kbits/s) and then try to minimize the delay.

To compute $\succcurlyeq_a$, we use the following definitions for $T(\vec{x})$ and $D(\vec{x})$. The throughput attainable is defined by the link in the path with the smallest allocated capacity, and

$$T(\vec{x}) = \min\{x_l \mid e_l \in P_a\}.$$

To define $D(\vec{x})$ we assume that VC a only transmits at rate γ_a. Any excess capacity allocated to VC a beyond its throughput goal is unused. Assuming an M/M/1 link queuing model, the worst case average delay on link i is given by

$$\frac{1}{C_l - S_l + (x_l - \gamma_a)}.$$

This assumes that all the remaining link supply $S_i - x_l$ is consumed by the throughput of other users of the link. So, $(C_i - S_i) + (x_i - \gamma_a)$ is the unused link capacity. Under the independence assumption [52],

$$D(\vec{x}) = \sum_{i \in P_a} \frac{1}{C_i - S_l + (x_l - \gamma_a)}.$$

At a given price system $\vec{p}$ we associate with each VC a a *demand set* of allocations $\Phi_a(\vec{p})$. We assume that VC a is endowed with a budget $W_a > 0$. The VCs' endowments are chosen by some policy external to the economy. Given a price system $\vec{p}$, the set of allocations affordable by VC a is

$$B_a(\vec{p}) = \{\vec{x} \mid \vec{p} \cdot \vec{x} \leq W_a\}.$$

This is called the *budget set* of VC a . The demand set is defined as the set of allocations in $B_a(\vec{p})$ that are optimally preferred by $\succcurlyeq_a$. That is,

$$\Phi_a(\vec{p}) = \{\vec{x} \mid \vec{x} \in B_a(\vec{p}), \text{ and if } \vec{y} \in B_a(\vec{p}) \text{ then } \vec{x} \succcurlyeq_a \vec{y}\}.$$

4.1.1 Computing the Demand Set

Virtual circuit a wants to send data from its source to its destination at rate γ_a. Since p_i is the cost per unit capacity of edge i, the total cost per transmission unit on a's path is

$$Q_a = \sum_{i \in P_a} p_i.$$

So, the total cost of achieving its throughput goal is $\gamma_a \cdot Q_a$, and VC a can achieve its throughput goal if $W_a \geq Q_a \cdot \gamma_a$. If this is not the case, VC a attempts to maximize its throughput subject to its monetary constraint. The following lemma states properties of $\Phi_a(\vec{p})$:

Lemma 1.1 : If $W_a \leq Q_a \cdot \gamma_a$ at prices $\vec{p}$, then:

1. $\Phi_a(\vec{p})$ is non-empty.
2. $\Phi_a(\vec{p})$ has a single element $\vec{\phi}^a$.
3. $\phi_i^a = \dfrac{W_a}{Q_a}$, if $i \in P_a$.
4. $\phi_i^a = 0$, if $i \notin P_a$.

This lemma is proved in the appendix. Stated simply, if $W_a \leq Q_a \cdot \gamma_a$, a VC simply divides its wealth over the total cost of the path.

If VC a can afford to meet its throughput goal, it purchases γ_a capacity of each edge in P_a and uses the remaining $W_a - (Q_a \cdot \gamma_a)$ to minimize the worst case average delay of its allocation. We ignore the edges not in P_a for the remainder of this section. Define $u_i = x_i - \gamma_a$, and $K_i = (C_i - S_i)$. The delay minimization problem is to minimize

$$F(\vec{u}) = \sum_{i \in P_a} \frac{1}{K_i + u_i}$$

subject to

1. $u_i \geq 0$.

2. $\vec{p} \cdot \vec{u} \leq W_a - Q_a \cdot \gamma_a$.

The set of feasible $\vec{u}$ is convex and compact, and the function F is continuous and strongly convex on this set. These facts are proved in the appendix. This implies that there exists a unique, feasible $\vec{u}^*$ minimizing $F(\vec{u})$[1] .

Formulating and simplifying the Kuhn-Tucker conditions for this minimization problem gives the following form for the optimal $\vec{u}^*$: For all i,

1. $u_i^* = 0$, or

2.

$$\frac{1}{(K_i + u_i^*)^2 \cdot p_i} = \lambda \tag{1}$$

where λ is the Lagrange multiplier. The first condition enforces the throughput goal $(u_i^* = x_i - \gamma_a \geq 0)$. For all links i with $x_i > \gamma_a$, the second condition states that the marginal decrease in delay per unit price is the same. If this were not the case, buying marginally less of one link and more of another would decrease delay without violating the budget constraint.

The algorithm for minimizing $D(\vec{x})$ computes λ and the links i with $u_i^* > 0$. For $u_i^* > 0$, equation (1) gives

$$u_i = \frac{1}{\sqrt{p_i \cdot \lambda}} - K_i.$$

The algorithm is a binary search on the possible values of λ. A VC agent uses the following algorithm to compute $\vec{u}^*$:

Algorithm 1 :

```
1. $\lambda^b = 0$ {The lower bound on $\lambda$}
2. {An upper bound on $\lambda$ from equation (1) and $u_i^* \geq 0$}
       $\lambda^u = \max \left\{ \frac{1}{(K_i)^2 \cdot p_i}, i \in P_a \right\}$
3. $\lambda = (\lambda^u - \lambda^b)/(2.0)$. {Test the midpoint of the range}
4. For $i \in P_a$ {Compute the $u_i$}:
    a.
       $u_i = \frac{1}{\sqrt{p_i \cdot \lambda}} - K_i.$
    b. If $u_i < 0$, $u_i = 0$. {Enforce throughput goal}
5. If $\vec{p} \cdot \vec{u} = W_a - Q_a \cdot y_a$, Stop, $\vec{u}$ is optimal.
6. If $\vec{p} \cdot \vec{u} > W_a - Q_a \cdot y_a$, $\lambda^b = \lambda$ . {Too many $u_i > 0$, increase $\lambda$ }
7. If $\vec{p} \cdot \vec{u} < W_a - Q_a \cdot y_a$, $\lambda^u = \lambda$ . {Too many $u_i = 0$, decrease $\lambda$ }
8. Go to 3 {Test new lambda}
```

For $p_i \geq \varepsilon = 0.01$ and $K_i = C_i - S_i = 0.8$, algorithm 1 computes λ with maximum error 0.001 in 25 iterations. This algorithm was implemented for the simulation study described in section 3 and this convergence behavior was confirmed.

After computing $\vec{u}^*$, VC a's demand set has a single element $\vec{\phi}^a$ with

1. $\phi_i^a = y_a + u_i^*$, if $i \in P_a$.
2. $\phi_i^a = 0$, if $i \notin P_a$.

4.1.2 Competitive Equilibrium and Price Updates

The total demand for a resource i at prices $\vec{p}$ is the sum of the individual demands, or

$$d_i(\vec{p}) = \sum_{a \in A} \phi_i^a(\vec{p}),$$

where $\phi_i^a(\vec{p})$ is VC a's demand for link i at prices $\vec{p}$. The *excess demand function* $\vec{Z}(\vec{p})$[40] represents the imbalance between supply and demand. Its i-th component is simply $Z_i(\vec{p}) = d_i(\vec{p}) - S_i$. The economy is in a *competitive equilibrium* at *equilibrium prices* $\vec{p}^*$ if $Z_i(\vec{p}^*) = 0$ for all i [40]. The flow control economy attempts to compute a competitive equilibrium. Equilibrium has the following properties: 1) No resources are unused. 2) Each VC's demand can be satisfied. Furthermore, we prove in section 2 that the equilibrium allocation is optimal.

For the flow control economy, this definition for competitive equilibrium is not adequate because S_i can be strictly greater than $d_i(\vec{p})$ no matter how low p_i falls. Figure 25 on page 88 depicts an example of this phenomenon. There are two VCs with $W_1 = W_2 = 1$ and $\gamma_1 = \gamma_2 = \infty$. All links have $S_i = 1$. By symmetry and lemma 1.1, the VCs evenly divide link 3, and VC 1 gets $< \frac{1}{2}, 0, \frac{1}{2} >$ and VC 2 gets $< 0, \frac{1}{2}, \frac{1}{2} >$. So, in equilibrium $\vec{Z}(\vec{p}^*) = < -\frac{1}{2}, -\frac{1}{2}, 0 >$.

To avoid this problem, we present a new definition of competitive equilibrium. The flow control economy is in equilibrium at prices $\vec{p}^*$ if for all i either: 1) $Z_i(\vec{p}^*) = 0$, or 2) $Z_i(\vec{p}^*) \leq 0$ and $p_i = \varepsilon$. This means that excess supply is allowed if the minimum price is charged for the resource.

Each data link i has a *supplier agent* that updates the p_i based on supply and demand. The algorithm for updating prices is a tatonement process and follows directly from the definition of equilibrium. The change in p_i is proportional to the imbalance between S_i and $d_i(\vec{p})$. The rule for updating p_i is:

$$p_i = \text{Max}\left[p_i + p_i \cdot \frac{Z_i(\vec{p})}{S_i}, \varepsilon\right].$$

The behavior of the Flow Control Economy as a whole is an iteration described by the following:

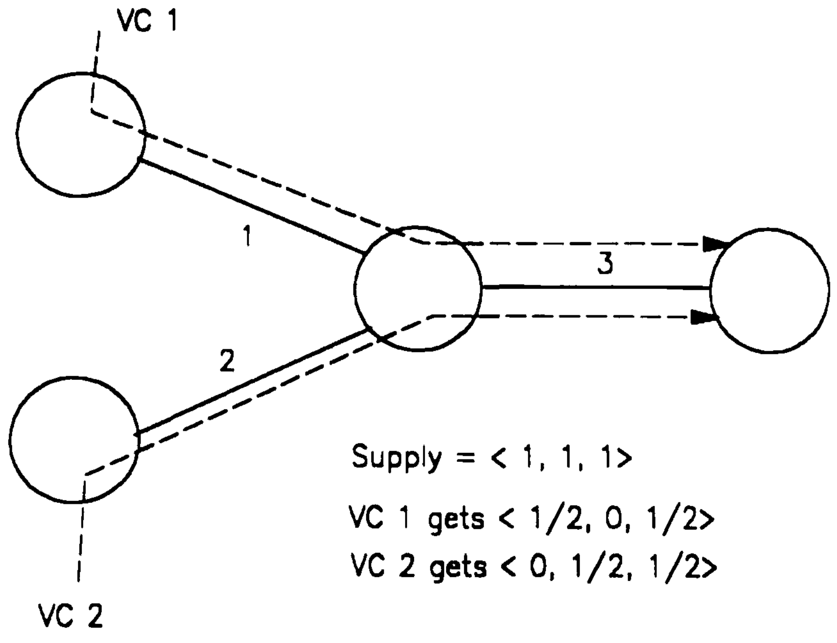


Figure 25. Supply Exceeds Demand at all Prices

Algorithm 2 :

1. Choose an initial price $\vec{p}$.

2. For all $a \in A$, compute $\vec{\phi}^a(\vec{p})$ {As describe in the preceding subsection}.

3. If for **all** $i \in E$, $(Z_i(\vec{p}) = 0)$ or $(Z_i(\vec{p}) \leq 0$ and $p_i = \varepsilon)$, an equilibrium has been reached and the **iteration stops.**

4. Otherwise, for all $i \in E$,

$$p_l = \text{Max}\left[p_l + p_l \bullet \frac{Z_l(\vec{p})}{S_l}, \varepsilon\right].$$

5. Go to 2.

In our implementation, we chose an initial price system with $p_i = \varepsilon$ for all i. The effects of the choice of an initial price system on the equilibrium computed is an open problem in our economy. In step 2, each VC computes its demand allocation at the current price system. To perform this computation, each VC requires information about the prices of resources on its path, and no other prices matter. In step 4, the supplier for link i needs to know the identifiers of the circuits using link i and their demands for link i in order to compute $Z_i(\vec{p})$. This means that the VCs can compute their demand vectors and the link suppliers can update prices using a decentralized algorithm as defined by Jaffe [49]. In the simple algorithm above, global information is required to detect equilibrium. We believe that there is a decentralized algorithm for equilibrium detection, but this is left for future work.

4.2 Optimality and Fairness

The goal of this section is to establish the degree to which the flow control economy accomplishes optimal and fair allocation of capacities. We proceed first with a formalized discussion of fair solutions to flow control problems.

A *flow control problem* is defined by:

1. A graph $G = \langle V,E \rangle$ with a capacity C_i for each $i \in E$. (As in section 2)
2. A set of virtual circuits $A = \{1, 2, \ldots, N\}$. VC $a \in A$ is characterized by:
    a. $P_a \subseteq E$: VC a's path.
    b. $\mathbf{X}^a$: Set of all possible allocations to a.
    c. An *Admissibility Correspondence*

$$F^a : \left[\prod_{(b \neq a) \wedge (b \in A)} \mathbf{X}^b \right] \rightarrow 2^{\mathbf{X}^a}.$$

For a given set of allocations to the other VCs, F^a defines the allocations that can be assigned to VC a.

d. A preference $\succcurlyeq_a$ on $\mathbf{X}^a$.

A *solution* to a flow control problem is a set of allocations $\{\vec{x}^1, \vec{x}^2, \dots, \vec{x}^N\}$ with the following property: For all $a \in A$

$$\vec{x}^a \in F^a(\{\vec{x}^1, \vec{x}^2, \dots, \vec{x}^N\} - \{\vec{x}^a\}).$$

The set of allocations is admissible with respect to each VC. For the flow control economy, the admissibility function is simply the budget constraint (and is not directly dependent on the allocations to other VCs).

The flow control problem and solution have been defined. We now present a set of norms that define *fair* solutions. The simplest condition is

(F1) A solution $\{\vec{x}^1, \vec{x}^2, \dots, \vec{x}^N\}$ to a flow control problem is fair if for all $a \in A$, $T(\vec{x}^a) > 0$.

This condition is not always satisfied by flow control algorithms that attempt to optimize a global performance metric. For example, in [10] it is shown that maximizing certain global metrics based on power may yield solutions that are not fair by F1. A stronger fairness condition is to allocate the same throughput to all VCs, but differing link capacities can make this impossible. Jaffe [48] proposed a fairness condition that takes differing link capacities into consideration. A link $i \in P_a$ is a *bottleneck link* for VC a if $T(\vec{x}^a) = x_i^a$. Link i constrains the throughput of VC a. Jaffe's fairness condition is

(F2) If link i is a bottleneck link of VC a, a solution $\{\vec{x}^1, \vec{x}^2, \dots, \vec{x}^N\}$ is fair if $x_i^a \geq x_i^b$ for all $b \in A$.

This means VC a should get at least as much of its bottleneck link as any other VC. F2 is based on the assumption that all VCs want to maximize throughput.

The third definition of fairness is related to a *Nash Equilibrium* [82].

(F3) A solution $\{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^N\}$ is fair if for all $a \in A$, $\vec{x}^a \succcurlyeq_a \vec{y}$ for all $\vec{y} \in F(\{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^N\} - \{\vec{x}^a\})$.

This simply states that a given allocation is fair if no VC can unilaterally improve its allocation given the allocations to the other VCs. This approach to fairness and optimality was proposed by Hsiao and Lazar [42, 43]. Unlike F1 and F2 which assume that all VCs are throughput oriented, F3 allows each VC to choose its individual preference relation.

F3 states that a solution is fair if no single VC can improve on its allocation. The fourth condition states that a solution is fair if no *group* of VCs can cooperatively improve their allocations. Define a *coalition* S as a subset of A. S can improve on solution $\{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^N\}$ if there is another solution $\{\vec{y}^1, \vec{y}^2, \ldots, \vec{y}^N\}$ with

1. $\vec{y}^a = \vec{x}^a$, $a \notin S$ (The non-members are not affected).
2. $\vec{y}^a \succcurlyeq_a \vec{x}^a$, $a \in S$ (No one in the coalition is worse off).
3. There exists $a \in S$ with $\vec{y}^a \succ_a \vec{x}^a$ (At least one member is strictly better off).

Given this definition,

(F4) A solution $\{\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^N\}$ is fair if no coalition can improve on their allocations.

Condition F4 is the definition of *Pareto-optimality*, and it also permits individual preferences.

The sets of fair solutions defined by these conditions are not disjoint. The following propositions state two obvious relations:

Proposition :

1. Condition F2 implies condition F1.
2. Condition F4 implies condition F3.

The fairness condition supplied by the flow control economy is based on the price system $\vec{p}$. The feasibility operator F^a is defined by the budget set. The following lemma shows that the flow control economy computes solutions that are fair by F1 and F3:

Lemma 2.1 : Let (1) $\vec{p}$ be a price system. (2) $\vec{\phi}^a(\vec{p})$ be VC a 's demand at $\vec{p}$. (3) $\vec{x} \neq \vec{\phi}^a(\vec{p})$ be any allocation. Then,

1. $\vec{\phi}^a(\vec{p}) \bullet \vec{p} = W_a$.
2. $\vec{x} \succcurlyeq_a \vec{\phi}^a(\vec{p})$ implies that $\vec{x} \bullet \vec{p} \geq \vec{\phi}^a(\vec{p}) \bullet \vec{p}$.
3. $\vec{x} \succ_a \vec{\phi}^a(\vec{p})$ implies that $\vec{x} \bullet \vec{p} > \vec{\phi}^a(\vec{p}) \bullet \vec{p}$.

The lemma is proved in the appendix.

By definition, the link prices and W_a are non-zero. The fairness of $\vec{\phi}^a(\vec{p})$ by condition F1 follows from part 1 of the lemma; Condition F3 follows from parts 2 and 3.

Lemma 2.1 can be used to prove that the flow control economy computes solutions that are fair by F4. The following theorem proves this assertion:

Theorem 2.1 : Let $\vec{p}$ be an equilibrium price in the flow control economy. Then, the set of allocations

$$\vec{\phi}^1(\vec{p}), \vec{\phi}^2(\vec{p}), \dots , \vec{\phi}^N(\vec{p})$$

is pareto-optimal and unique.

Proof :

(unique) Each $\vec{\phi}^i(\vec{p})$ is unique, so the aggregation of the choices in unique.

(pareto-optimality) Assume a coalition $S = \{1, 2, \dots , s\}$ chooses a set of allocation with

1. $\vec{\phi}'^i(\vec{p}) \succcurlyeq_i \vec{\phi}^i(\vec{p})$, $\forall i \in S$.
2. $\vec{\phi}'^j(\vec{p}) \succ_j \vec{\phi}^j(\vec{p})$, for at least $1\ j \in S$.

We know from lemma 2.1 that

1. $\vec{\phi}'^{i}(\vec{p}) \cdot \vec{p} \ge \vec{\phi}^{i}(\vec{p}) \cdot \vec{p}, \ \forall i \in S$.
2. $\vec{\phi}'^{j}(\vec{p}) \cdot \vec{p} > \vec{\phi}^{j}(\vec{p}) \cdot \vec{p}$, for j.

So,

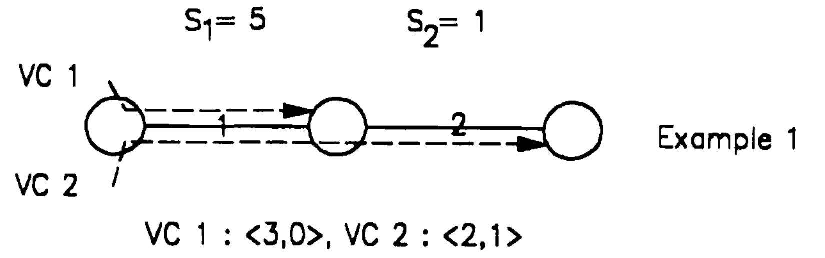
$$\sum_{i=1}^{s} \vec{\phi}'^{i}(\vec{p}) \cdot \vec{p} > \sum_{i=1}^{s} \vec{\phi}^{i}(\vec{p}) \cdot \vec{p} = \sum_{i=1}^{s} W_i .$$

The coalition cannot afford the better set of allocations at prices $\vec{p}$.

This proof only relies on the properties of the preference relation established in lemma 2.1. Theorem 2.1 is not specific to the preference relation of section 1. Any other preference relation possessing these properties could be added to the flow control economy without decreasing the fairness properties or Pareto-optimality of the equilibrium allocation.

The equilibrium allocation is optimal because an allocation that improves the lot of any VCs forces unfair allocations on others. This depends on the assumption that the tatonement process sets prices that accurately reflect the relative importance of the data links.

The examples in Figure 26 on page 94 show the relationship between conditions F2 and F4. In example 1, assume VC 1 and 2 use only throughput for their preference ($\gamma_a = \infty$), and consider the allocations $< 3, 0 >$ to VC 1 and $< 2, 1 >$ to VC 2. VC 2 can increase VC 1's throughput by giving it one unit of capacity on link 1. This does not decrease VC 2's throughput because $S_2 = 1$. So, the allocation is not Pareto-optimal. It is fair by definition F2 because VC 1 gets more of link 1 than VC 2. Jaffe's algorithm computes a solution that is more fair than he claims. The solution is in fact fair by F4. Other definitions of fairness equivalent to F2 include special rules to ensure that the solution is also Pareto-optimal [9]. The main advantage of the flow control economy over this previous work is that diverse preferences are allowed. The previous work assumes that all VCs are throughput oriented.



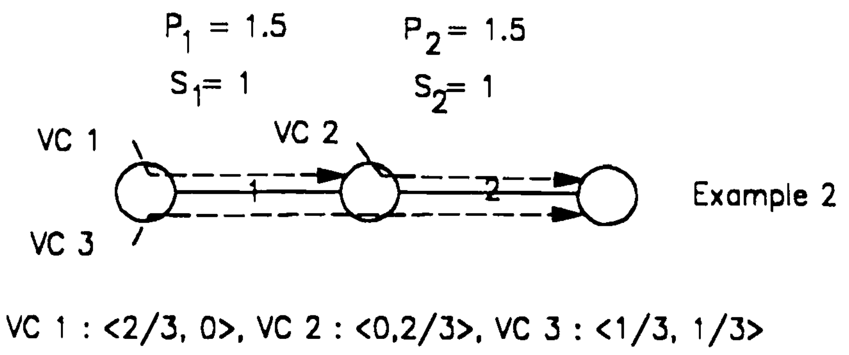


Figure 26. Fairness Conditions F2 and F4

Example 2 in Figure 26 on page 94 shows that the flow control economy computes solutions that are not fair by F2. In this example, for $i = 1, 2, 3$, $y_i = 1$, and $W_i = 1$. At prices $\vec{p} = < 1.5, 1.5 >$, the economy is in equilibrium with allocations $< \frac{2}{3}, 0 >$ to VC 1, $< 0, \frac{2}{3} >$ to VC 2, and $< \frac{1}{3}, \frac{1}{3} >$ to VC 3. VC 3 is bottlenecked on links 1 and 2 and has less capacity than the other VCs. This example shows that the flow control economy can be unfair to long VCs under definition F2. An alternative argument can state that condition F2 is unfair to short VCs. Under F2, the total amount of capacity assigned to a VC increases with its length. This decreases the amount assigned to shorter VCs. External policies can cause the flow control economy to reach a mean between these conflicting goals by the amount of money assigned to long VCs compared to short VCs.

4.3 Existence of Equilibrium

In this section, we show that a competitive equilibrium always exists, and that there may be infinitely many equilibrium price vectors. The traditional approach in economic theory for proving the existence of an equilibrium price vector is based on fixed point theorems [1, 5, 40]. A point $\vec{p}^0$ is a fixed point of a function $\vec{f}$ if $f(\vec{p}^0) = \vec{p}^0$. A function $\vec{f}(\vec{p}) = \vec{p} + \vec{Z}(\vec{p})$ is defined and shown to have a fixed point $\vec{p}^0$. This implies that $\vec{Z}(\vec{p}^0) = 0$ and $\vec{p}^0$ is an equilibrium price vector.

We must modify these tools to prove existence of equilibrium in the flow control economy because of the non-standard definition of equilibrium that may have $\vec{Z}(\vec{p}^*) \neq \vec{0}$ in equilibrium. Economic theory also assumes that the demand functions of the individual agents are continuous with $\vec{p}$. We cannot make this assumption, and we prove that the demand function defined in section 1 is continuous.

We define a new function $\vec{H}(\vec{p})$ with the property that $\vec{H}(\vec{p}) = \vec{0}$ if, and only if, $\vec{p}$ is an equilibrium price system. We then show that $\vec{f}(\vec{p}) = \vec{p} + \vec{H}(\vec{p})$ has a fixed point.

We define the function $\vec{H}(\vec{p})$ with i-th component

$$H_i(\vec{p}) = (p_i - \varepsilon) \cdot Z_i(\vec{p}) + Z_i(\vec{p}) + |Z_i(\vec{p})|.$$

By our definition of equilibrium, if $\vec{p}^*$ is an equilibrium price vector, then for every i

1. $Z_i(\vec{p}^*) = 0$, or
2. $p_i = \varepsilon$ and $Z_i(\vec{p}^*) \leq 0$.

If $Z_i(\vec{p}^*) = 0$, then $H_i(\vec{p}^*) = 0$. If $p_i = \varepsilon$, then the first term in $H_i(\vec{p}^*)$ is 0. If $Z_i(\vec{p}^*) \leq 0$, then the second and third terms cancel and $H_i(\vec{p}^*) = 0$. So, if $\vec{p}^*$ is an equilibrium price, $H_i(\vec{p}^*) = 0$ for all i, which implies $\vec{H}(\vec{p}^*) = \vec{0}$.

Assume that $\vec{H}(\vec{p}) = \vec{0}$. We must show that $\vec{p}$ is an equilibrium price vector. We know by definition that $p_i - \varepsilon \geq 0$. So, $H_i(\vec{p}) = 0$ implies that $Z_i(\vec{p}) \leq 0$. This, in turn, implies that

$$H_i(\vec{p}) = (p_i - \varepsilon) \cdot Z_i(\vec{p}).$$

This term is 0 only if $Z_i(\vec{p}) = 0$ or $p_i = \varepsilon$. So, $H_i(\vec{p}) = 0$ implies that $Z_i(\vec{p}) = 0$ or $Z_i(\vec{p}) \leq 0$ and $p_i = \varepsilon$. The following theorem has been shown.

Theorem 3.1 : $\vec{H}(\vec{p}) = \vec{0}$ if, and only if, $\vec{p}$ is an equilibrium price vector.

Define the function $\vec{f}(\vec{p}) = \vec{p} + \vec{H}(\vec{p})$. We apply the following theorem to show that $\vec{f}$ has a fixed point.

(Brouwer's fixed point) **Theorem :** Let X be a compact, convex subset of $\mathbb{R}^M$, and let $\vec{f}$ be a continuous function mapping X into X. Then there exists $\vec{x}^0 \in X$ such that $\vec{f}(\vec{x}^0) = \vec{x}^0$.

Brouwer's theorem is discussed in economic contexts in [1, 5, 40].

To apply the fixed point theorem, we must

1. Define the set X.
2. Prove $\vec{f}$ maps X into itself.
3. Prove continuity of $\vec{f}$ on X.

Let $W_T = \sum_{i=1}^{N} W_i$. This is the total amount of money possessed by the VCs. Define the set X as

$$X = \left\{ \vec{x} \mid \varepsilon \leq x_i \leq \frac{2W_T}{\varepsilon} \right\}.$$

X is simply a cube in $\mathbb{R}^M$ and contains its boundary. So, it is is compact and convex.

We must show that $f(\vec{p}) \in X$ for all $\vec{p} \in X$. To do so, we make two assumptions that do not limit generality. First, we assume that $0 \leq S_i \leq 1$ for all i. If this is not the case, an equivalent economy can be created by dividing all S_i by Max $\{S_i \mid 1 \leq i \leq M\}$. Secondly, we assume that $\varepsilon \leq S_i$ for all i. This is possible because the only constraint is $\varepsilon > 0$.

Since $p_i \geq \varepsilon > 0$, f_i takes its minimal value at $\vec{p}^0$ when $Z_i(\vec{p})^0$ is minimal. Our assumptions that $S_i \leq 1$ implies that $Z_i(\vec{p}) \geq -1$, and we have

$$f_i(\vec{p}^0) \geq p_i^0 + (\vec{p}_i^0 - \varepsilon) \bullet (-1) + (-1) + |-1| = \varepsilon.$$

To prove that $f_i(\vec{p}) \le \frac{2W_T}{\varepsilon}$ we define a function

$$g(p_l) = p_l + (p_l - \varepsilon)\frac{W_T}{p_l} + \frac{2W_T}{p_l}.$$

The budget constraint implies that $Z_i(\vec{p}^0) \le \frac{W_T}{p_i^0} - S_i$. We know that $S_i > 0$ and we have $g(p_i^0) \ge f_i(p_i^0)$. The second derivative of $g(\vec{p})$ is

$$g''(\vec{p}) = \frac{2(2-\varepsilon)W_T}{p_l^3},$$

which is strictly positive on $[\varepsilon, \frac{2W_T}{\varepsilon}]$ provided that $\varepsilon < 2$. So, $g(p_i)$ takes its maximal value at either $p_i = \varepsilon$ or $p_i = \frac{2W_T}{\varepsilon}$. At $p_i = \varepsilon$,

$$f_l(\vec{p}^0) \le \varepsilon + \frac{2W_T}{\varepsilon} - 2S_l \le \frac{2W_T}{\varepsilon}.$$

At $p_i^0 = \frac{2W_T}{\varepsilon}$, $Z_i(\vec{p}^0) \le 0$ because of the budget constraint and

$$f_l(\vec{p}^0) \le \frac{2W_T}{\varepsilon}.$$

This shows that for all $\vec{p} \in X$, $\varepsilon \le f_i(\vec{p}) \le \frac{2W_T}{\varepsilon}$ for all i which means that $\vec{f}(\vec{p}) \in X$. So, $\vec{f}$ maps X into itself.

The function $\vec{H}(\vec{p})$ is continuous if the excess demand function $\vec{Z}(\vec{p})$ is continuous. The excess demand function is simply

$$\vec{Z}(\vec{p}) = \left(\sum_{i=1}^{N} \vec{\phi}^i(\vec{p})\right) - \vec{S},$$

where $\vec{\phi}^a(\vec{p})$ is VC a's demand at $\vec{p}$ and $\vec{S}$ is the vector of link supplies. We will show that $\vec{\phi}^a(\vec{p})$ is continuous, which implies continuity of $\vec{Z}$.

Lemma 1.1 shows that for VC a, if

$$\gamma_a \sum_{j \in P_a} p_j \geq W_a$$

the agent's demand is a continuous function of $\vec{p}$. If the agent can afford its throughput goal, and attempts to minimize delay, we have to show that the $\vec{u}^*$ solving the minimization problem in the previous section is a continuous function of $\vec{p}$.

Let $U(\vec{p})$ be defined as follows: $\vec{u} \in \mathbb{R}^M$ is in $U(\vec{p})$ if,

1. $u_i \geq 0$

2.

$$\sum_{j \in P_a} (p_l \cdot u_l) \leq W_a - (\gamma_a) \sum_{j \in P_a} p_l.$$

Define the function $F: U(\vec{p}) \rightarrow \mathbb{R}$ as

$$F(\vec{u}) = \sum_{i \in P_a} \frac{1}{(K_i + u_i)}$$

as in section 2.

The following lemmas are needed to prove continuity and are proved in this chapter's appendix.

Lemmas 3 :

1. For any $\vec{p}$, F is *continuous* and *strongly convex* on $U(\vec{p})$.
2. For any $\vec{p}$, $U(\vec{p})$ is *compact* and *convex*.
3. A sequence of points $\{\vec{x}^m\}$ in $\mathbb{R}^N$ converges to $\vec{x}^0$ if, and only if, every sub sequence $\{\vec{x}^{m_k}\}$ contains a subsequence $\{\vec{x}^{m'_k}\}$ that converges to $\vec{x}^0$.

We prove the following theorem:

Theorem 3.2 : For all $\vec{p}$, let $\vec{x} = M(\vec{p})$ denote a vector in $U(\vec{p})$ minimizing $F(\vec{x})$. Then $M(\vec{p})$ is a continuous function of $\vec{p}$.

Proof:

1. $M(\vec{p})$ is a *function:*
   - a. $U(\vec{p})$ is compact, and F is continuous on $U(\vec{p})$, so there exists an $\vec{x} \in U(\vec{p})$ minimizing F [1].
   - b. So $M(\vec{p})$ is defined for all $\vec{p}$.
   - c. Furthermore, *convexity* of $U(\vec{p})$ and *strong convexity* of F on $U(\vec{p})$ imply that $\vec{x}$ is *unique* for any $\vec{p}$.
   - d. So, M is not multivalued and is a function.
2. $M(\vec{p})$ is *continuous*:
   - a. Pick a point $\vec{p}^0$. Let $\vec{p}^m$ be a sequence converging to $\vec{p}^0$.
   - b. Denote $\vec{x}^m = M(\vec{p}^m)$ and $\vec{x}^0 = M(\vec{p}^0)$.
   - c. We must show that as $\vec{p}^m \to \vec{p}^0$, $\vec{x}^m \to \vec{x}^0$.
   - d. Define two sequences $\vec{w}^m$ and $\vec{u}^m$ with the following properties:
     1) $\vec{w}^m$:
        - a) $\vec{w}^m \in U(\vec{p}^m)$.
        - b) For all $\vec{z} \in U(\vec{p}^m)$,

          $$||\vec{w}^m - \vec{x}^0|| \leq ||\vec{z} - \vec{x}^0||.$$

          $\vec{w}^m$ is the point in $U(\vec{p}^m)$ closest to $\vec{x}^0$.
     2) $\vec{u}^m$:
        - a) $\vec{u}^m \in U(\vec{p}^0)$.

b) For all $\vec{y} \in U(\vec{p}^0)$,

$$||\vec{u}^m - \vec{x}^m|| \leq ||\vec{y} - \vec{x}^m||.$$

$\vec{u}^m$ is the point in $U(\vec{p}^0)$ closest to $\vec{x}^m$.

$\vec{u}^m$ and $\vec{w}^m$ exist because $U(\vec{p}^m)$ and $U(\vec{p}^0)$ are compact. Figure 27 on page 101 shows an example of the sequences $\vec{x}^m$, $\vec{w}^m$, $\vec{u}^m$ and their relationship to $\vec{p}^0$ and $\vec{x}^0$ in the two dimensional case.

e. As $\vec{p}^m \to \vec{p}^0$,

1) $||\vec{w}^m - \vec{x}^0|| \to 0$

2) $||\vec{u}^m - \vec{x}^m|| \to 0$.

This is intuitively clear. It is formally proved in an appendix to this chapter.

f. Let $\vec{u}^{m_k}$ be a subsequence of $\vec{u}^m$. Since $U(\vec{p}^0)$ is compact, $\vec{u}^{m_k}$ has a convergent subsequence $\vec{u}^{m'_k} \to \vec{u}^0 \in U(\vec{p}^0)$.

g. Assume that $\vec{u}^0 \neq \vec{x}^0$.

h. By construction $||\vec{u}^m - \vec{x}^m|| \to 0$. By f., $\vec{u}^{m'_k} \to \vec{u}^0$. So, $\vec{x}^{m'_k} \to \vec{u}^0$.

i. $\vec{w}^m \to \vec{x}^0$ which implies that $\vec{w}^{m'_k} \to \vec{x}^0$.

j. By definition, $F(\vec{w}^{m'_k}) \geq F(\vec{x}^{m'_k})$.

k. By the assumption that $\vec{u}^0 \neq \vec{x}^0$ and uniqueness of the optimal, $F(\vec{u}^0) > F(\vec{x}^0)$.

l. By continuity of F, $F(\vec{x}^{m'_k}) \to F(\vec{u}^0)$, and $F(\vec{w}^{m'_k}) \to F(\vec{x}^0) < F(\vec{u}^0)$. However, $F(\vec{w}^{m'_k}) \geq F(\vec{x}^{m'_k})$ implies that $F(\vec{x}^0) \geq F(\vec{u}^0)$ which contradicts $F(\vec{x}^0) < F(\vec{u}^0)$

m. So, $\vec{u}^0 = \vec{x}^0$ and every subsequence of $\{\vec{u}^m\}$ contains a subsequence that converges to to $\vec{x}^0$, which by the lemma above means $\vec{u}^m \to \vec{x}^0$.

n. $||\vec{u}^m - \vec{x}^m|| \to 0$, and by m., $\vec{x}^m \to \vec{x}^0$.

3. This proves the theorem.

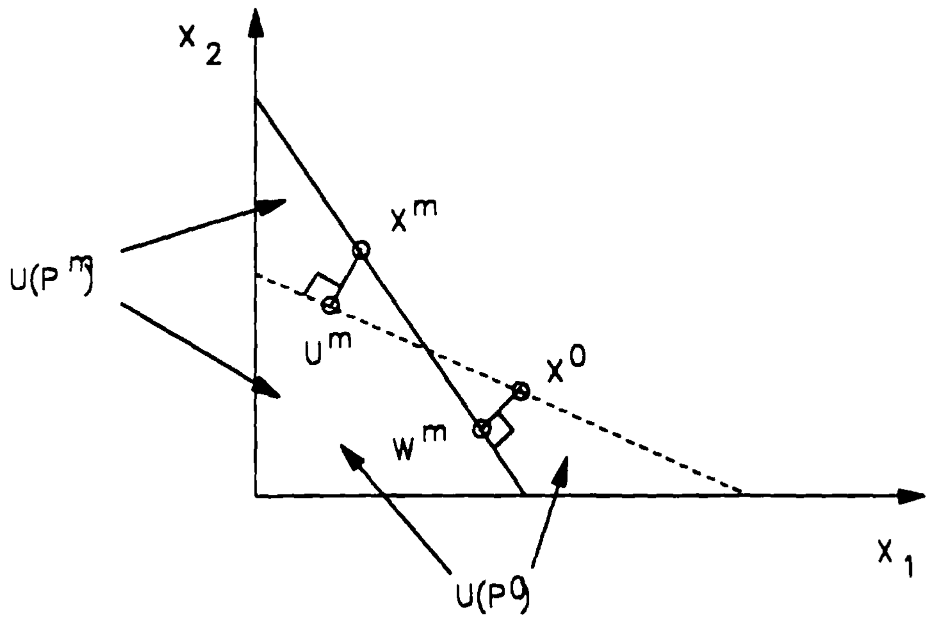


Figure 27. The two dimensional example: $\vec{u}^m$ is the point in $U(\vec{p}^0)$ closest to $\vec{x}^m$. $\vec{w}^m$ is the point in $U(\vec{p}^m)$ to $\vec{x}^0$.

Lemma 1.1 shows that $\vec{\phi}^a(\vec{p})$ is continuous at any $\vec{p}$ with $\gamma \sum_{i \in P_a} p_i > W_a$. Theorem 3.2 shows that $\vec{\phi}^a(\vec{p})$ is continuous at $\vec{p}$ when $\gamma \sum_{a \in P_a} p_i \leq W_a$. When $\gamma \sum_{i \in P_a} = W_a$, the feasible vector minimizing F is $\vec{0}$. This implies that $\vec{\phi}^a(\vec{p})$ is continuous when $\gamma \sum_{i \in P_a} = W_a$. So, the VCs demand is continuous at all prices.

The demand functions of the individual VCs are continuous, which means that $\vec{H}$ is continuous. Brouwer's fixed point theorem can be applied to prove that $f(\vec{p}) = \vec{p} + \vec{H}(\vec{p})$ has a fixed point. This proves the main result of this section stated by the following theorem.

Theorem 3.3 : There is an equilibrium price vector $\vec{p}^*$ for the flow control economy.

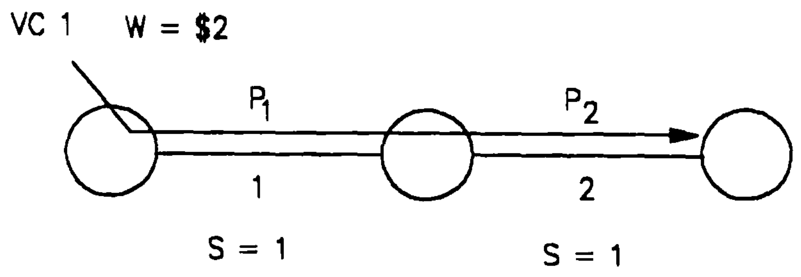


Figure 28. Infinite Equilibria: If $p_1 + p_2 = 2$, the economy is in equilibrium.

The theorem states that there is at least 1 equilibrium price vector. The following example demonstrates that there may be infinitely many equilibrium price vectors. Figure 28 on page 103 depicts the example. There are two links with $S_1 = S_2 = 1$ and one VC using the links. The VC has $W = \$2$ and $\gamma = \infty$. At any price vector with $p_1 + p_2 = 2$, the VC demands $< 1, 1 >$ by lemma 1.1. In this case, supply equals demand for both links and the economy is in equilibrium. So, there are infinitely many equilibrium prices.

This example illustrates that there are infinitely many equilibrium *prices*. However, in this example there is only 1 equilibrium allocation. We have not yet determined if all equilibria prices yield the same allocations, or if there are multiple allocations as well. This is left for future work.

4.4 Convergence to Equilibrium

This section presents the results of experiments to determine the convergence behavior of the economy (Algorithm 2). In general, the tatonement process in an economy does not always converge to an equilibrium [80]. Economists have developed algorithms for computing equilibrium price vectors, but these algorithms are extremely complicated, have high computational overhead and are inherently centralized [80]. We conducted several hundred simulation experiments on algorithm 2, and the flow control economy always converged. Proof of convergence or finding a non-converging example is left for future work.

We implemented the flow control economy and simulated its behavior on randomly generated networks. In each simulation, the network had 50 virtual circuits and 25 data links. The capacity of link i, C_i, was a random variable with uniform distribution on the interval $[1, 5]$. S_i was $0.8 \times C_i$ for all links.

Figure 29 on page 104 and Figure 30 on page 105 plot the convergence to equilibrium of the flow control economy for various network definition parameters. The X-axis represents the number of iterations of the economy (the number of price updates in step 4 of algorithm 2). The Y-axis represents the relative distance from the eventual equilibrium allocation of resources. Denote $\vec{d}(\vec{p}^k)$ as the total demand vector for all agents at price $\vec{p}^k$ of iteration k; $\vec{d}(\vec{p}^*)$ is the total demand in equilibrium. In this notation, the Y-axis represents

$$\frac{||\vec{d}(\vec{p}^*) - \vec{d}(\vec{p}^k)||}{||\vec{d}(\vec{p}^*)||}.$$

The Y-axis represents the normalized error between $\vec{d}(\vec{p}^k)$ and $\vec{d}(p^*)$.

In Figure 29 on page 104, the average convergence for three sets of random economies is presented. These economies differ by the probability of link i being in agent a's path. If this probability is denoted $Pr\{i \in P_a\}$, the three curves plot average convergence for random economies with $Pr\{i \in P_a\} = 0.1, 0.2,$ and 0.3 for all i and a. These values define random economies in which the mean length of a virtual circuit is 2.5, 5.0, 7.5, respectively. The first conclusion drawn from this graph is that the economy converges very rapidly. After 15 iterations, all three economies are within

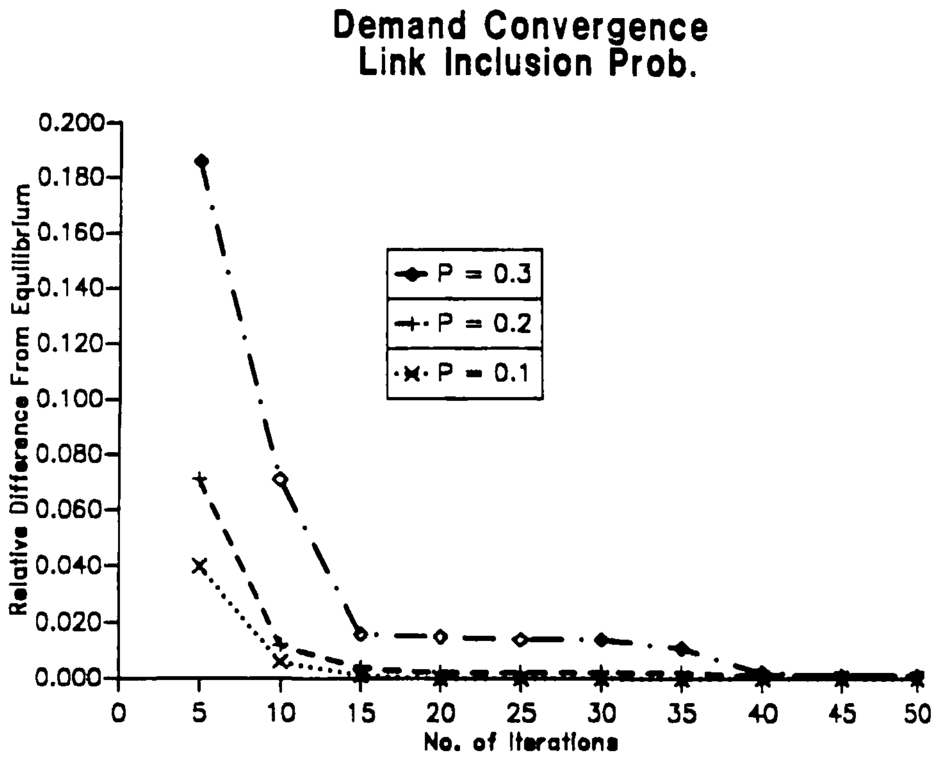


Figure 29. Convergence to Equilibrium Demand: This figure plots the economy's convergence to equilibrium. The three curves represent random economies in which the probability of link i being in VC *a*'s path is 0.1, 0.2, and 0.3. These yield mean VC lengths of 2.5, 5 and 7.5 links respectively. The mean VC throughput demand was 1.0.

2% of the equilibrium allocation. The second conclusion is that the mean VC length has little effect on the rate of convergence. The explanation for this fact is given below.

Figure 30 on page 105 plots the convergence behavior of the economy for various VC throughput goals. In this experiment, each virtual circuit's throughput demand γ_a was a random variable uniformly distributed on the interval $[0,T]$. The three curves represent the cases of T = 0.5, 2.0, and 3.0. In these economies, the mean VC length was 5 links. The same conclusion can be drawn from these experiments as from the previous. The economy converges rapidly and after 20 iterations all three economies are within 2% of the equilibrium allocation. Varying the mean throughput goal has little effect on the convergence behavior of the economy.

The flow control economy converges very rapidly to the equilibrium allocation of capacity to agents. The economy takes many more iterations to identify that an equilibrium has been com-

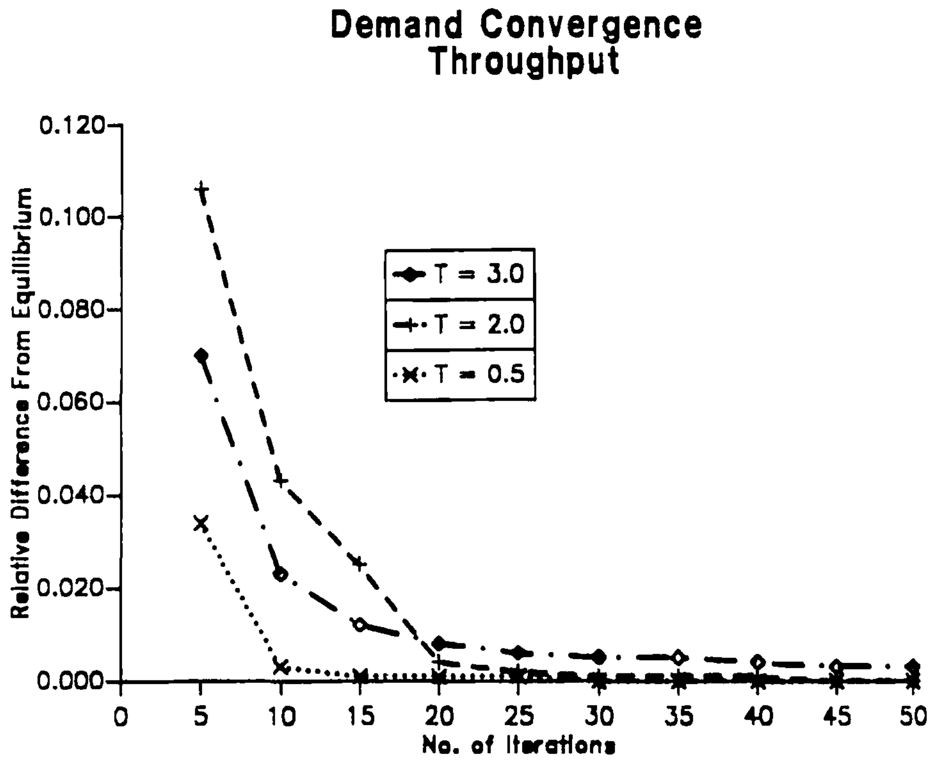


Figure 30. Convergence to Equilibrium Demand: This figure plots the economy's convergence to equilibrium. The three curves represent random economies in which the mean VC throughput demand was uniformly distributed in the interval [0, *T*], where T = 0.5, 2.0, and 3.0. The mean VC length was 5 links.

puted, however. In the simple economy of Figure 31 on page 106, there are two VCs (VC 1 and VC 2). $P_1 = 1,3$ and $P_2 = 2,3$, and both VCs have $W = 1$ and $\gamma = 1$. The supply vector is $\vec{S} = <1, 1, 1.98>$. By symmetry and lemma 1.1, the equilibrium allocation for this economy is:

1. VC 1 gets $<0.99, 0, 0.99>$.
2. VC 2 gets $<0, 0.99, 0.99>$.

In the simulation of this simple economy, the minimum price was $\varepsilon = 0.001$. The economy required over 1000 iterations to detect that it had computed an equilibrium. The economy *did* converge very rapidly to the equilibrium, but did not realize that it had. After only four iterations,

$$\frac{||\vec{d}(\vec{p}^4) - \vec{d}(\vec{p}^*)||}{||\vec{d}(\vec{p}^*)||} = 0.002.$$

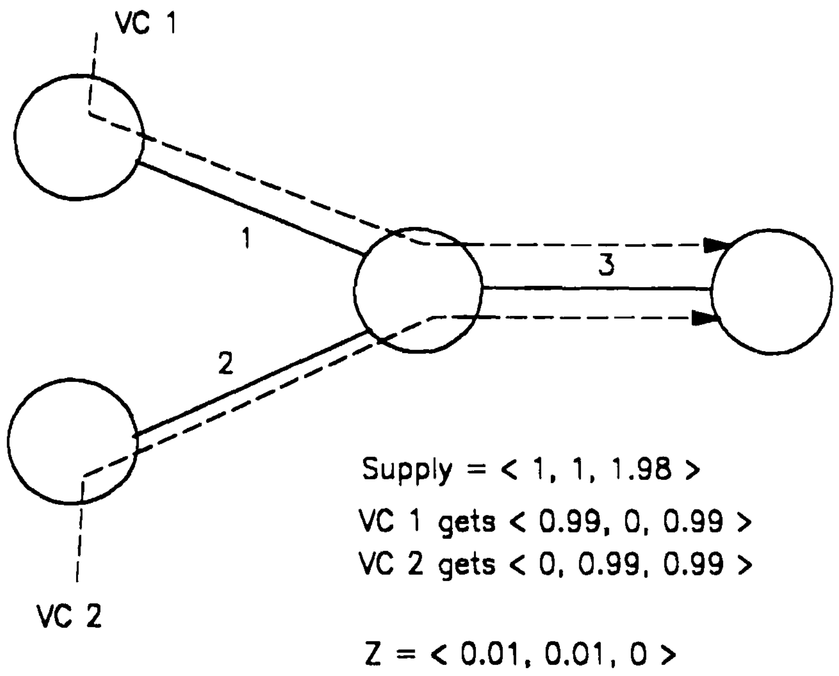


Figure 31. Convergence detection problem.

The detection problem is caused by the fact that in equilibrium some resource may have demand slightly less than supply. In this case, the price of the resource is slowly driven down to ε . Let i be the most utilized link whose equilibrium demand is less than supply. That is, the i that minimizes:

$$\Delta = \frac{(S_i - d_i(\vec{p}))}{S_i}$$

and has $p_i^* = \varepsilon$. Suppose that the equilibrium allocation for this link is computed in iteration L. The economy must drive the price of edge i to ε. The price p_i^K in iteration $K > L$ is simply

$$p_l^L \cdot (1 - \Delta)^{(K-L)}.$$

For $p_l^K \leq \varepsilon$, we have

$$K = L + \frac{\log \varepsilon - \log p_l^L}{\log(1 - \Delta)}. \tag{2}$$

As $\Delta \to 0$, the value of K in equation (2) goes to infinity. Factors such as mean VC throughput goal and mean VC length only indirectly effect the detection of equilibrium. The detection of equilibrium is solely determined by the most utilized, non-saturated link.

There are heuristic solutions to the equilibrium detection problem. One approach is to arbitrarily stop the economy after some threshold K iterations. Our experiments suggest that $K = 25$ yields allocations within 2% of optimal for realistically large networks. Another approach is to monitor the change in demand in successive iterations. In this case, the economy stops iterating if

$$\frac{||\vec{d}(\vec{p}^{k+1}) - \vec{d}(\vec{p}^{k})||}{||\vec{d}(\vec{p}^{k})||} \leq \delta,$$

where δ is a parameter. We implemented and tested this second heuristic for the economies of Figure 29 on page 104, Figure 30 on page 105 and Figure 31 on page 106, setting $\delta = 0.01$. The heuristic detects equilibrium in the pathological economy of Figure 31 on page 106 after 2 iterations. The relative error in the demand vector when the heuristic guesses that equilibrium was found was 0.002. For the six sets of random economies of Figure 29 on page 104 and Figure 30 on page 105, on the average the heuristic guesses that equilibrium has been found after 10.1 iterations. The average relative error if the economy stopped when equilibrium was guessed was 0.021.

4.4.1 Network Updates

The simulation experiments in the preceding discussion demonstrated that the flow control economy can rapidly compute an equilibrium allocation of resources when starting from an initial state in which no resources have been allocated. In this section, we study the convergence of the economy to a new equilibrium when a VC leaves the economy, or when a new VC is started up.

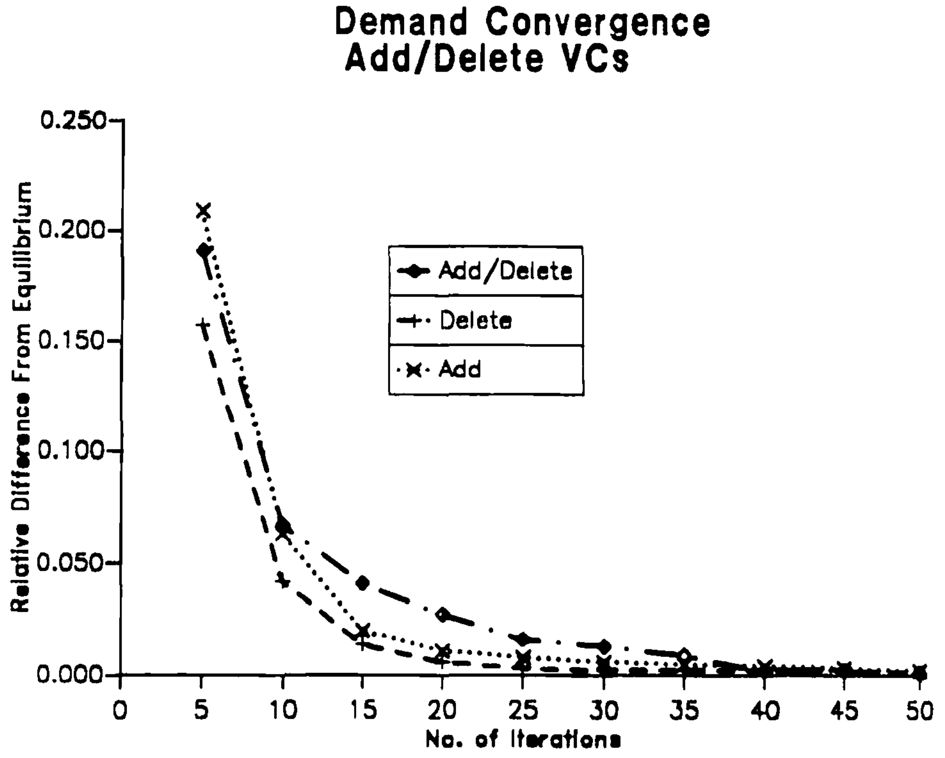


Figure 32. Network Updates: This figure plots the average convergence of the economy to the new equilibrium after a new VC is activated or a VC terminates. The three curves represent adding 50 VCs 1 at a time, deleting 50 VCs and a random ordering of 50 adds and deletes.

Figure 32 on page 109 contains the results of simulation experiments testing the convergence. In this figure, a network with 25 links and 50 random VCs is generated and the equilibrium is computed. Then a VC addition or deletion is simulated. After each change, the new equilibrium is computed. The curves represent the average number of iterations required to reach the new equilibrium. The three cases are: 1) Adding 50 VCs, 2) Deleting 50 VCs, and 3) a random ordering of 50 additions and deletions.

Three main conclusions can be drawn from these experiments. First, the economy rapidly computes the new equilibrium after an update. The economy is within 5% of the new equilibrium after 15 iterations in all cases. The second conclusion is that moving from one equilibrium to the new equilibrium after an update requires approximately as many iterations as the computation of the

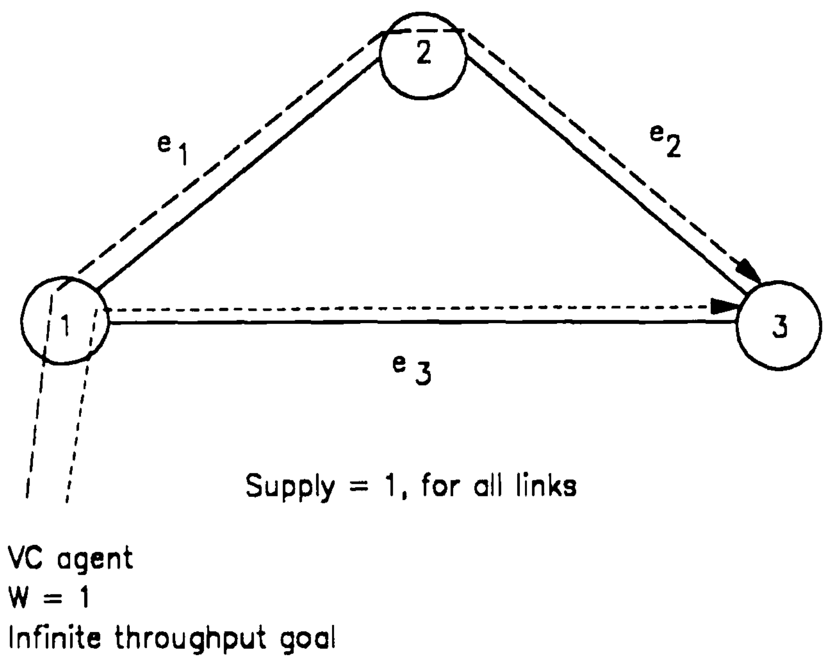


Figure 33. Routing Pathology: If the VC agent can choose a new path after each price update, the economy does not converge.

initial equilibrium. This is due to the same phenomenon described above. The third conclusions is that the flow control economy can effectively deal with dynamic network environments.

In the simulations in this section, the path chosen for a new VC is determined before the economy starts iterating to a new equilibrium. The price information can help the VC agent choose the "best" path. However, the VC cannot change its routing decision after the tatonement process begins. That is, dynamic routing after each price update cannot be allowed. If this is allowed, there are cases in which an equilibrium will never be computed.

Figure 33 on page 110 depicts an example of this problem. In this figure, there are three links e_1, e_2 and e_3. Each link has $S_i = 1$ and initial price $p_i = \varepsilon$, and we assume $\varepsilon < 0.5$. There is a single VC agent that wants to route traffic from node 1 to node 3. This agent has throughput goal $\gamma = \infty$ and wealth $W = \$1$. The agent can choose either path $\{e_1, e_2\}$ or simply $\{e_3\}$. The agent choose the cheaper path $\{e_3\}$ and demands capacity $\frac{1}{\varepsilon}$ which is greater than S_3. So, p_3 increases.

As soon as $p_3 > 2 \cdot \varepsilon$, the agent switches to the cheaper path $\{e_1, e_2\}$ and demands capacity $\frac{1}{2 \cdot \varepsilon}$ of each link. This causes the price of p_3 to fall because demand is now 0, and causes and the prices of p_1 and p_2 to rise. As soon as $p_1 + p_2 > p_3$, the VC switches back to path $\{e_3\}$. So, the VC oscillates between paths $\{e_3\}$ and $\{e_1, e_2\}$. The economy never reaches an equilibrium.

4.5 Comparison with Related Work

The flow control problem for virtual circuit networks has been extensively studied. Comprehensive surveys of this work can be found in [9, 37, 81]. In this section, we compare previous work to the flow control economy. Flow control has two goals. The first is obtaining an optimal trade-off between throughput and delay [9]. The second is fairly allocating the communication resources to the set of VCs. The economy is compared to previous work with respect to meeting these goals.

Window flow control is the most common flow control algorithm used in practice [9, 81]. Each VC has a source A that sends data units and a destination B that accepts the data units and sends acknowledgements back to A. Flow control is implemented by assigning a window size W to A. Source A is allowed to have at most W unacknowledged data units sent to B. When the window is full, A must wait until receiving an acknowledgement before sending any additional data.

The idea behind window flow control is that when the network is becoming congested, the delays of routing a data unit from A to B and routing the acknowledgement back increase. This decreases the rate at which A can transmit. This occurs over all VCs, and the system wide average delay is improved by decreasing the throughputs of the senders.

The main advantage of window flow control relative to the flow control economy is that window flow control is dynamic. At most W data units can be transmitted before throughput is decreased to compensate for a transient period of congestion [9]. The flow control economy computes static transmission rates and does not react to transient conditions. A second advantage is that the window sizes can be chosen to guarantee that buffer overflows do not occur at any routing nodes.

One problem with window flow control is choosing the window sizes. The throughput of a VC can partially be determined by the choice of its window size, e.g. - large windows increase throughput. However, the throughput and delay of the VC are also determined by the window sizes of other VCs. This makes the choice of window sizes complicated. The flow control economy has a similar problem with the initial endowment of wealth to VCs. Exploring how the choice of the W_a's effects the relative throughputs and delays of the VCs is left for future work.

Another problem with window flow control is that it implements a system wide throughput-delay trade-off. For example, the set of VCs experiencing delays due to congestion may be throughput oriented, and should not have their window sizes decreased. The flow control economy does not have this problem.

Window flow control can be unfair to short VCs [9]. The window size of a VC must be an increasing function of the number of links in its path to meet its throughput goals during periods with light contention. During periods of congestion, this makes it likely that a long VC gets more capacity of a heavily loaded link than a shorter VC sharing the same link [9]. The fairness of the flow control economy with respect to long VCs versus short VCs was described in section 2.

The flow control problem can be combined with circuit routing to form a single optimization problem [9]. The problem is to choose a path P_a and a transmission rate r_a for each VC a that minimizes:

$$\sum_a D_a + \sum_a e_a(r_a).$$

D_a is VC a's delay given the routing and transmission rates. $e_a(r_a)$ is a *penalty function* for the VC. As r_a decreases, this function increases and defines the penalty for flow controlling this circuit.

The main disadvantage to this approach is the it requires computationally intensive, centralized computing to determine the routing and transmission rates. The flow control economy also performs a minimization problem when the VCs try to minimize delay. We have a efficient algorithm that is specific to our optimization problem. It is not clear how the computational overheads of the two approaches compare.

A second disadvantage of the combined routing-flow control problem is that the optimization must be recomputed each time a new VC is activated or an existing VC terminates. This incurs overhead. Furthermore, the new paths may be different from the existing paths. So, additional overhead is incurred terminating the existing paths and starting the new ones.

The definition of the penalty functions adds complexity to the flow control problem. e_a must reflect VC a's need for throughput. The rate actually assigned to VC a also depends on the penalty functions of other VCs. So, the interactions between the penalty functions increase problem complexity. The assignment of wealth to a VC in the flow control economy also should consider the values assigned to other VCs. This is straight forward, however, and does not require comparisons of throughput-delay goals.

Finally, combining routing and flow control in a single optimization problem may assign 0 throughput to some VCs. So, it is not fair be condition F1.

The main advantage of the combined flow control-routing approach is that it can improve on the allocations given to the VCs compared to the flow control economy. We have proven that the economy computes pareto-optimal resource allocations given a fixed set of VC paths. However, choosing an alternate set of paths may result in an allocation the improves the lot of all VCs. We demonstrated that the economy cannot implement dynamic routing in addition to flow control. This is a major shortcoming of the economy. Quantifying the performance degradation due to not dynamically assigning routes is an area for future work.

A third class of flow control algorithms are *Bottleneck* or *max-min* algorithms [9]. These algorithms are based input rate adjustment. The algorithms' primary goals is to compute allocations which are fair by condition F2. An efficient decentralized algorithm for computing rate assignments that are fair by condition F2 has been proposed by Jaffe [48]. Other algorithms are presented in [9].

The main disadvantage of bottleneck flow control is the assumption that all VCs are throughput oriented. This approach is less general than the flow control economy. We compared the fairness properties of the economy with bottleneck flow control in section 1.

A major benefit of the flow control economy is that it supports decentralized choice of individual throughput-delay trade-offs. Some previous work on the flow control problem has allowed a VC to define its throughput-delay goals independently of other VCs. Sanders [78, 79] presented an algorithm for computing optimal transmission rate allocations under a model in which each VC has an individual *utility function* that defines its throughput-delay trade-off. Possible utility functions were not defined. Sanders algorithm is decentralized and asynchronous, and maximizes the sum of the individual utilities. The algorithm can assign transmission rate 0 to a circuit and is unfair by condition F1. The algorithm also requires some coordination between choice of utility functions.

Bharath-Kumar and Jaffe [10] proposed a generalized form of power for for representing throughput-delay trade-offs. This work focused on maximizing a global objective function based on the power obtained by individual VCs, e.g. - the sum of the individual powers. The individual VCs did not choose their own definition of power and were assumed to be homogeneous, however. In some examples, their algorithm computes rate allocations that are a Nash Equilibrium. It was shown in section 2 that a Nash Equilibrium is not Pareto-optimal. In their work, Bharath-Kumar and Jaffe observed that their algorithm computed an allocation that could be improved upon by all VCs.

The related flow control work described in chapter 2 and that of Bharath-Kumar and Jaffe use power to model throughput-delay trade offs. Let γ_k be VC k's throughput, and D_k is its delay. VC k chooses a constant β_k and attempts to maximize its power

$$\frac{(\gamma_k)^{\beta_k}}{D_k}.$$

Power is an awkward representation for a individual VC throughput-delay goals. It would be very difficult for the user of VC k to choose the constant β_k that achieves its throughput-delay goals. The model used in the flow control economy makes specifying throughput-delay goals simpler.

4.6 Conclusions

In this chapter we have applied concepts from mathematical economics to develop an effective flow control mechanism. The immediate advantages of our algorithm compared to previous flow control mechanisms, are that: 1) it captures in a precise manner both diversity of priorities among virtual circuits in a network as well as diversity of their performance objectives (throughput vs. delay). 2) it accomplishes resource allocations that are optimally fair (in a formal sense).

Our algorithm is decentralized and is simple to implement. Additionally, simulation studies demonstrated that the economy efficiently computes optimal resource allocations for realistically large networks. We showed that the economy rapidly computes the new optimal allocation when a new virtual circuit is activated or an existing circuit terminates. Finally, we demonstrated the limitations of the economy as a dynamic, virtual circuit routing algorithm through a simple example.

In this chapter, we demonstrated the benefits of applying economic theory to resource management problems in distributed systems. Pareto-optimality was shown to provide a powerful new definition for optimal, fair resource allocations in heterogeneous network environments. Economic techniques were used to prove optimality of the competitive equilibrium computed by the economy, and that an equilibrium does exist for all networks under our model. This chapter illustrated that two main challenges must be overcome when applying economic theory to resource management in distributed computer systems. First, the underlying economic models must be modified to accurately reflect the problem being studied. Secondly, and most importantly, it must proven that the system possesses fundamental properties assumed by economic theory, and it cannot be assumed that the economy possesses these properties. For example, continuity of demand with respect to prices is assumed in economic theory. In this chapter we had to prove that the agents' diverse demand functions were continuous.

Several areas for future work remain. First, two theoretical questions were raised. One is uniqueness of the equilibrium allocation. The second is a proof of convergence or a counter example. Another area is expanding the economy to incorporate other models of VC goals. This would increase the generality of the results. Finally, the use of link prices to determine long term routing decisions, to guide capacity planning and to set demand based rate setting in public access networks are potentially fruitful areas for further work.

4.7 Proof of Lemmas

This section proves lemmas on which results in this chapter depend. Let $G = <V, E>$ be the graph of the underlying network, and let the current price vector be $\vec{p} = <p_1, p_2, \ldots, p_m>$ with $0 < \varepsilon \le p_i < \infty$ for all i. A VC agent a in this economy has a non-empty path $P_a = \{l_{a,1}, l_{a,2}, \ldots, l_{a,m(a)}\}$, wealth $W_a > 0$ with which it purchases allocations and a throughput goal γ_a. To make the notation simpler, the $\vec{p}$ will be dropped from the allocation vectors when the context is clear, i.e. - $\vec{\phi}(\vec{p})$ is written simply $\vec{\phi}$.

For the proof of lemma 1.1, define a special allocation $\vec{\beta}^a$ for VC a with:

1. $\beta_i^a = 0$, if $i \notin P_a$.
2. $\beta_i^a = \frac{W_a}{Q_a}$, if $i \in P_a$.

where $Q_a = \sum_{i \in P_a} p_i$.

Lemma 1.1 : If $W_a \leq Q_a \bullet \gamma_a$ at prices $\vec{p}$, then:

1. $\Phi_a(\vec{p})$ is non-empty.
2. $\Phi_a(\vec{p})$ has a single element $\vec{\phi}$.
3. $\phi_i = \frac{W_a}{Q_a}$, if $i \in P_a$.
4. $\phi_i = 0$, if $i \notin P_a$.

Proof:

All four propositions will be proven if we can show $\vec{\beta}^a$ is VC a's unique optimal choice at prices $\vec{p}$.

First,

$$\vec{\beta}^a = \sum_{i \in P_a} p_i \bullet \frac{W_a}{Q_a} = \frac{W_a}{Q_a} \bullet Q_a = W_a.$$

So $\vec{\beta}^a$ is affordable and $B_a(\vec{p})$ is not empty. The preference relation $\succcurlyeq_a$ is a partial order, and has a maximal element. This means that $\Phi_a(\vec{p})$ is not empty.

Suppose there exists another allocation $\vec{\alpha}^a \succcurlyeq \vec{\beta}^a$. Then we have the following:

1. $\alpha_i^a \geq 0$ implies that $\alpha_i^a \geq \beta_i^a$ for $i \notin P_a$.
2. $T(\vec{\alpha}^a) \geq T(\vec{\beta}^a)$, which in turn implies $\alpha_i^a \geq \frac{W_a}{Q_a}$, $i \in P_a$.

If $\vec{\alpha}^a \sim_a \vec{\beta}^a$, then $T(\vec{\alpha}^a) = T(\vec{\alpha}^a) = \frac{W_a}{Q_a}$ and $\vec{\alpha}^a = \vec{\beta}^a$. If $\vec{\alpha^a} >_a \vec{\beta^a}$, then $T(\vec{\alpha}^a) > T(\vec{\alpha}^a) = \frac{W_a}{Q_a}$. This means $\alpha_i^a > \frac{W_a}{Q_a}$, and $\vec{\alpha}^a \bullet \vec{p} > W_a$. So, $\vec{\alpha}^a$ is not affordable.

Lemma 2.1 : Let (1) $\vec{p}$ be a price system. (2) $\vec{\phi}^a(\vec{p})$ be agent a's demand at $\vec{p}$. (3) $\vec{x} \neq \vec{\phi}^a(\vec{p})$ be any allocation. Then,

1. $\vec{\phi}^a(\vec{p}) \bullet \vec{p} = W_a$.
2. $\vec{x} \succcurlyeq_a \vec{\phi}^a(\vec{p})$ implies that $\vec{x} \bullet \vec{p} \geq \vec{\phi}^a(\vec{p}) \bullet \vec{p}$.
3. $\vec{x} >_a \vec{\phi}^a(\vec{p})$ implies $\vec{x} \bullet \vec{p} > \vec{\phi}^a(\vec{p}) \bullet \vec{p}$.

Proof:

If $T(\vec{\phi}^a(\vec{p})) \leq \gamma_a$ the results are obvious. So, assume that $T(\vec{\phi}^a(\vec{p})) \geq \gamma_a$.

(1) If $\vec{\phi}^a(\vec{p}) \bullet \vec{p} < W_a$, then VC a has a surplus of $\delta = W_a - \vec{\phi}^a(\vec{p}) \bullet \vec{p}$, with $\delta > 0$. Buying $\frac{\delta}{p_i}$ of any link $i \in P_a$ improves the delay because delay decreases monotonically as a function of x_i when $x_i \geq \gamma_a$. So, $\vec{\phi}^a(\vec{p})$ was not optimal, and this is a contradiction.

(2,3) Assume the opposite: $\vec{x} \succcurlyeq_a \vec{\phi}^a(\vec{p})$, and $\vec{x} \bullet \vec{p} < \vec{\phi}^a(\vec{p}) \bullet \vec{p}$. Perform the following transformations

1. $u_i = \phi_i^a(\vec{p}) - \gamma_a$, for $i \in P_a$.
2. $w_i = x_i - \gamma_a$, for $i \in P_a$.
3. $u_i = \phi_i^a(\vec{p})$, $i \notin P_a$.
4. $w_i = x_i$, $i \notin P_a$.

This means that $\vec{p} \bullet \vec{w} \leq W_a - Q_a \times \gamma_a$, and $\vec{w}$ is feasible (affordable). $\vec{x} \sim_a \vec{\phi}^a(\vec{p})$ means that in the minimization problem of section 2, $F(\vec{u}) = F(\vec{w})$. This contradicts the strong convexity of F. If $\vec{x} >_a \vec{\phi}^a(\vec{p})$, then $F(\vec{w}) < F(\vec{u})$. This contradicts the optimality of $\vec{u}$ and the definition of $\vec{\phi}^a(\vec{p})$.

lemma 3.1 : $F(\vec{u})$ is continuous and *strongly convex*.

Proof: :

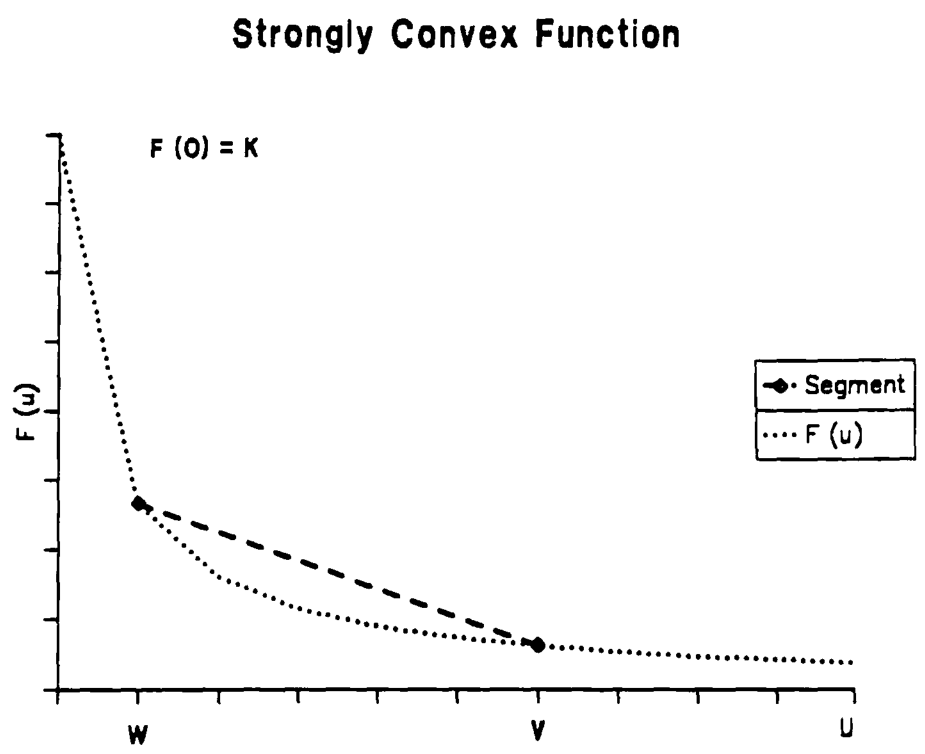


Figure 34. Strong convexity of F_i

F is the sum of m functions

$$F_i(\vec{u}) = \frac{1}{K_i + u_i}$$

each of which is strongly convex and continuous ($K_i > 0, u_i \geq 0$). Figure 34 on page 118 shows the form of F_i. It is clear that the segment connecting $F_i(w)$ and $F_i(v)$ always lies above the curve.

Strong convexity of the F_i's means that for all $u, w \in U$,

$$\alpha \cdot F_i(\vec{u}) + (1 - \alpha) \cdot F_i(\vec{w}) > F(\alpha \cdot \vec{u} + (1 - \alpha) \cdot \vec{w}).$$

This in turn implies,

$$\sum_{i \in P_a} \alpha \cdot F_i(\vec{u}) + (1-\alpha) \cdot F_i(\vec{w}) > \sum_{i \in P_a} F_i(\alpha \cdot \vec{u} + (1-\alpha) \cdot \vec{w}).$$

This means $\alpha \cdot F(\vec{u}) + (1-\alpha)F(\vec{w}) > F(\alpha \cdot \vec{u} + (1-\alpha) \cdot \vec{w})$ and F is strongly convex.

Lemma 3.2 : U is convex and compact.

Proof :

We re-write $W_a - C_a \cdot \gamma_a$ as K_a to simplify the notations.

(Compact) : U is compact if it is bounded and closed. By definition, U is bounded below by $\vec{0} = <0, 0, \dots, \dots, 0>$. We know that $\vec{p} \cdot \vec{u} \leq K_a$. This requires that $p_i \cdot u_i \leq K_a$ which in turn means that $u_i \leq \frac{K_a}{p_i}$. We know that $p_i > 0$ which means that $\frac{K_a}{p_i} < \infty$. So, U is bounded from above by the vector

$$< \frac{K_a}{p_1}, \frac{K_a}{p_2}, \dots, \frac{K_a}{p_M} >.$$

For U to be closed, any sequence of vectors $\vec{u}^n \in U$ that converges must converge to an element $\vec{u}^*$ of U. Assume the sequence $\vec{u}^n$, $n = 1, 2, \dots$ converges to $\vec{u}^*$. This means that for all $i = 1, 2, \dots, M$, u_i^n converges to u_i^*. Assume that $u_i^* < 0$. This means that $u_i^* = 0 - \delta$, $\delta > 0$. By definition, $u_i^n \geq 0$, for all n. This means that $\lim |u_i^n - u_i^*| \geq \delta$. This contradicts the assumption of convergence, and $u_i^* \geq 0$, $i = 1, 2, \dots, M$.

$|\vec{p} \cdot \vec{u}^n - \vec{p} \cdot \vec{u}^*| =$

$$\sum_{i=1}^{M} | u_i^n - u_i^* | \cdot p_i.$$

We know that as $n \rightarrow \infty$, $|u_i^n - u_i^*| \rightarrow 0$ that in turn implies $|\vec{p} \cdot \vec{u}^n - \vec{p} \cdot \vec{u}^*| \rightarrow 0$. We know that $\vec{p} \cdot \vec{u}^n \leq K_a$, so $\vec{p} \cdot \vec{u}^* \leq K_a$.

We have shown that $u_i^* \geq 0$ and $p \cdot \vec{u}^* \leq K_a$. So, $\vec{u}^* \in U$ and U is closed. U is closed and bounded which means that it is compact.

(Convex) : For U to be convex, for any two vectors $u, v \in U$ and any α, $0 \leq \alpha \leq 1$, the vector

$$\vec{w} = \alpha \cdot \vec{u} + (1 - \alpha)\vec{v}$$

must be in U. By definition, $u_i \geq 0$, $v_i \geq 0$, $\alpha \geq_0$ and $1 - \alpha \geq 0$, so $w_i = \alpha \cdot u_i + (1 - \alpha) \cdot w_i \geq 0$.

$\vec{p} \cdot \vec{w} = \vec{p} \cdot (\alpha \cdot \vec{u}) + \vec{p} \cdot ((1 - \alpha) \cdot \vec{v})$. This in turn means, $\vec{p} \cdot \vec{w} = \alpha \cdot (\vec{p} \cdot \vec{u}) + (1 - \alpha) \cdot (\vec{p} \cdot \vec{v})$. By definition, $\vec{p} \cdot \vec{u} \leq K_a$ and $\vec{p} \cdot \vec{v} \leq K_a$. So, $\vec{p} \cdot \vec{w} \leq \alpha \cdot K_a + (1 - \alpha) \cdot K_a = K_a$.

Lemma 3.3 : : A sequence of points $\{\vec{x}^m\}$ in $\mathbb{R}^N$ converges to $\vec{x}^0$ if, and only if, every sub sequence $\{\vec{x}^{m_k}\}$ contains a subsequence $\{\vec{x}^{m'_k}\}$ that converges to $\vec{x}^0$.

Proof :

The if part follows directly from the definition of convergence

For the only if part, assume that $\vec{x}^m \not\rightarrow \vec{x}^0$. Also assume that every subsequence contains a subsequence converging to $\vec{x}^0$.

$\vec{x} \not\rightarrow \vec{x}^0$ implies that there exists a $\varepsilon > 0$ such that for any $k \geq 1$, there is an m_k such that $||\vec{x}^{m_k} - \vec{x}^0|| > \varepsilon$. For $k = 1, 2, \ldots$ take the sequence of points $\vec{x}^{m_k}$. By definition, this sequence cannot contain a subsequence converging to $\vec{x}^0$ because for all k, $||\vec{x}^{m_k} - \vec{x}^0|| > \varepsilon$.

This contradicts the assumptions, and the lemma is shown.

We now prove one of the assertions made in the proof of continuity of a VC's demand function with respect to prices. This was theorem 4.2. Let $\vec{x}^0, \vec{p}^0, \vec{p}^m$ be defined as for theorem 4.2. For the sequence of points $\vec{p}^m \rightarrow \vec{p}^0$, define a sequence of points $\vec{w}^m \in U(\vec{p}^m)$ where

$$||\vec{x}^0 - \vec{w}^m|| \leq ||\vec{x}^0 - \vec{z}^m||$$

for all $\vec{z} \in U(\vec{p}^m)$. We prove the following lemma:

Lemma 3.4 : As $\vec{p}^m \rightarrow \vec{p}^0$, $||\vec{x}^0 - \vec{w}^m|| \rightarrow 0$.

Proof :

We construct a sequence $\vec{y}^m$ that converges to $\vec{x}^0$. This will prove convergence of $\vec{w}^m$.

If $\vec{x}^0 = \vec{0}$ then $\vec{y}^m = \vec{0}$.

Assume that $\vec{x}^0$ is a vector in $U(\vec{p}^0)$. Let $S \subseteq \{1, 2, \ldots, M\}$ be the links for which $\vec{x}_i^0 > 0$. Define e_m as

$$e_m = \min \{(p_s^0 - p_s^m) \mid s \in S\}.$$

We have $e_m \rightarrow 0$.

We know that

$$p_s^m + e_m \leq p_s^0. \tag{1}$$

The budget constraint tells us that

$$\sum_{i=1}^{M} p_i^0 x_i^0 \leq W - \gamma \sum_{i=1}^{M} p_i.$$

Plugging the inequality (1) into the budget constraint gives

$$\sum_{s \in S} (p_s^m + e_m) x_s^0 \leq W - \gamma (\sum_{i=1}^{N} p_i^m) - \gamma e_m M.$$

Rearranging terms gives

$$\sum_{s\in S} p_s^m x_s^0 + e_m \sum_{s\in S} x_s^0 + \gamma e_m M \le W - \gamma \sum_{l=1}^{M} p_l^m .$$

We multiply and divide the second term in this inequality by the cardinality of S, denoted $|S|$, and get

$$\sum_{s\in S} p_s^m x_s^0 + |S| e_m \left(\frac{\left(\sum_{s\in S} x_s^0 \right) + \gamma M}{|S|} \right) \le W - \gamma \sum_{l=1}^{M} p_l^m . \tag{2}$$

For simplicity of notation, let

$$A = \frac{\left(\sum_{s\in S} x_s^0 \right) + \gamma M}{|S|} .$$

By definition of the cardinality of a set we have

$$|S| e_M A = \sum_{s\in S} e_M A ,$$

and multiplying and dividing each i-th term p_i^m gives

$$|S| e_M A = \sum_{s\in S} p_s^m \left(\frac{e_M A}{p_s^m} \right) .$$

This means we can rewrite inequality (2) as

$$\sum_{s\in S} p_s^m\left(x_s^0 + \frac{e_m A}{p_s^m}\right) \leq W - y\sum_{l=1}^{M} p_l^m. \tag{3}$$

We know that $e_m \rightarrow 0$, and $p_s^m \rightarrow p_s^0$. By definition, $p_s^0 > 0$, and by the definition of the set S, we know $x_s^0 > 0$. This implies that we can find $K \in \mathbb{Z}$ such that for all $s \in S$, if $m > K$,

$$x_s^0 \geq -\frac{e_m}{p_s^m} \bullet \frac{A}{|S|}. \tag{4}$$

The sequence $\{\vec{y}^m\}$ is defined by components as follows:

1. $y_i^m = 0$, if $m \leq K$

2.

$$y_i^m = x_i^0 + \frac{e_m}{p_s^m} \bullet \frac{A}{|S|}, \text{ if } m > K \text{ and } i \in S$$

3. $y_i^m = 0$, if $m > K$ and $i \notin S$

From inequality (4), we know that $y_i^m \geq 0$, and inequality (3) shows that $\vec{y}^m$ is affordable. So, $\vec{y}^m \in U(\vec{p}^m)$.

We know that:

1. $e_m \rightarrow 0$.

2. $p_i^m \rightarrow p_i^0 > 0$.

3. $|S| > 0$.

This means that $y_i^m \rightarrow x_i^m$. This proves the lemma

5.0 The Data Management Economy

This chapter presents an economy that manages data objects in a distributed system. The economy's goal is to improve performance by regulating the number of copies of each object, and the assignment of copies to processors. The effectiveness of the economy is evaluated by a simulation based performance study.

The main results of this section is a set of simulation based performance experiments that demonstrates that the data management economy substantially improves performance compared to static global policies. The improvement is achieved in two ways. First, the number of copies of each data object is controlled to reflect the ratio of read access vs. write access. When an object is written, all copies must be updated. If the write access rate is high, there should be few copies to avoid update overhead. If read access is predominant, there should be many copies. This increases the probability that a read can be processed locally or at a nearby processor. The second way in which performance is improved is by assigning the copies to processors based on localities in transaction reference patterns. This increases the probability of a data access be satisfied locally or at a nearby processor. Both of these improvements are achieved by decentralized, competitive decision making.

This chapter is structured as follows. Section 1 describes the distributed system and defines the data management problem. Section 2 presents the data management economy and a running example. The main results of this chapter are contained in performance experiments described in section 3. The data management economy solves a general version of the File Allocation Problem. Previous work on this problem is compared with the economy in section 4. Section 5 contains conclusions and directions for future work.

5.1 Problem Statement and System Model

The distributed system consists of N processing nodes $P_1, P_2, \dots, P_N$. Each P_i has a processing parameter r_i that defines the rate at which it can process operations on local data. The nodes are connected by a set of communication links $E = \{e_{ij}\}$, and link e_{ij} has delay d_{ij}.

There are M data objects in the system denoted $D_1, D_2, \dots, D_M$. $S(D_i)$ defines the size of D_i in bytes. The problem definition treats the data objects as abstract resources. In a real system, they could correspond to *Relations* [20, 21], *tuples*, files, records or any other data structure. There may be multiple copies of each data object at distinct processor nodes.

Transactions enter the system at all processors. A transaction is a sequence of *Read* and *Write* operations on the data objects. We assume that a transaction's operations must be serviced sequentially. Operation *Read*(D_i) can be performed on any copy of D_i. If a transaction submits *Write*(D_i, *value*), all copies are updated.

The data management problem is to minimize mean transaction response time. The parameters that can be controlled are: 1) The number of copies of each data object. 2) The assignment of copies to processing nodes.

Our model does not explicitly study the effects of concurrency control [8]. The data management economy is compatible with concurrency control protocols that do block transactions, e.g. - Timestamp, Optimistic [7, 8]. Expanding the model to include lock based concurrency control is an area for future work.

5.2 The Economy

There are four types of agents in the data management economy. These are *transactions*, *processors*, *data object managers* and *local data agents*. A transaction T that enters the system at node P_i pays P_i to perform its data access operations. T is endowed with an initial allocation of money M_T.

Processor node P_i sells data access to local transactions. To do so, P_i *leases* copies of data objects from the data object managers. If transaction T submits a *Read*(D_j) or a *Write*(D_j, *value*) and P_i does not lease a copy of object D_j, the work is *subcontracted* to some processor with a copy. A processor's sole goal is to maximize profit.

Each data object is controlled by a data object manager that leases copies of the object to processors. The data object manager for D_j is also responsible for updating all copies when a *Write*(D_j, *value*) occurs. A data object manager's goal is to maximize profit.

There is one local data agent at each node in the system. The local data agent at P_i acts as an intermediary between P_i and the object managers, which may reside at other nodes. Figure 35 on page 127 shows a simple data management economy that is used as an example in this chapter. There are three processors in this economy (P_1, P_2, P_3) and three objects (A, B, C). There may be multiple read copies of an object, but only one write copy exists. In the example, P_1 has a read (triangle) and write (square) copy of A, and P_1 and P_2 have a read copy. The meaning of the prices is described below.

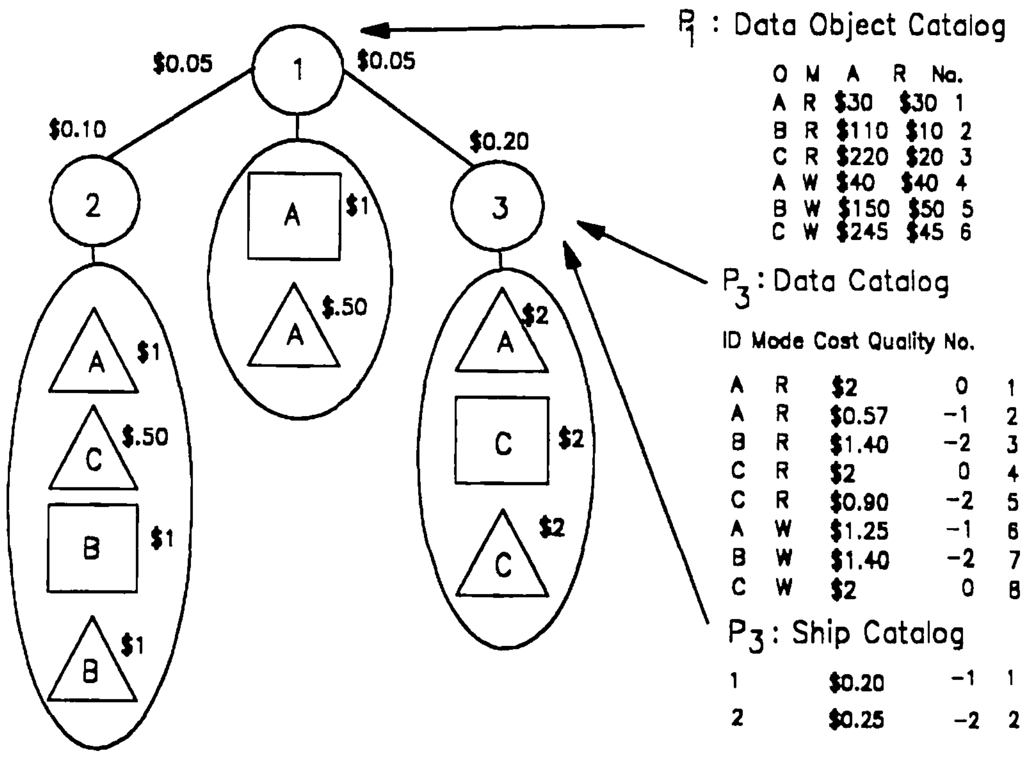
5.2.1 Processor Agent

A processor node P_i sells data access to local transactions. In Figure 35 on page 127, processor P_3 has a read and write copy of object C and can sell *Read*(C) and *Write*(C, *value*) to transactions. For a *Read*(C), P_3 collects its fee and returns the value of C to the transaction. For a *Write*(C, *value*), P_3 collects the fee and updates the local copy of C. P_3 also informs C's data object manager of the update. The manager updates all read copies of C at other processors (described below), and returns a result to P_3. At this point, the *Write*(C,*value*) is complete and the transaction can submit the next operation.

The processors also sell data access to each other through a *contract* protocol. In Figure 35 on page 127, processor P_3 sends an *advertisement* to processor P_1 informing P_1 that P_3 can sell read/write access to object C and the prices it charges. P_1 can then supply these services to local transactions. P_1 can also supply *Read*(C) and *Write*(C, *value*) to other nodes, and sends an advertisement to P_2.

If a transaction T at P_2 purchases *Write*(C, *value*), P_2 *subcontracts* the work to P_1. The subcontract includes payment to P_1 for the write and the data written. P_1 in turn subcontracts the access to P_3. P_3 processes the *Write*(C, *value*) exactly as a local transaction's write. The fee is collected, the local and remote copies are updated, and a response is sent to P_1. P_1 then responds to P_2, and T can submit its next operation.

Processor P_i independently sets the prices for the services it supplies. The price P_i sets for a service is based on any costs to P_i of supplying the service, and the demand for the service. In the example, P_3 charges \$2 for *Write*(C, *value*). This price is based on the *Write*(C, *value*) demand from local transactions and P_1, and on the lease price of the write copy. P_3 charges \$0.20 per message for link e_{31}. So, the total cost P_3 charges P_1 for *Write*(C, *value*) is \$2.20. P_1 sets its fee for *Write*(C, *value*)



P_1 : Data Object Catalog

O	M	A	R	No.
A	R	$30	$30	1
B	R	$110	$10	2
C	R	$220	$20	3
A	W	$40	$40	4
B	W	$150	$50	5
C	W	$245	$45	6

P_3 : Data Catalog

ID	Mode	Cost	Quality	No.
A	R	$2	0	1
A	R	$0.57	−1	2
B	R	$1.40	−2	3
C	R	$2	0	4
C	R	$0.90	−2	5
A	W	$1.25	−1	6
B	W	$1.40	−2	7
C	W	$2	0	8

P_3 : Ship Catalog

1	$0.20	−1	1
2	$0.25	−2	2

Figure 35. The Data Management Economy: This figure depicts an example of the data management economy. There are three processors P_1, P_2 and P_3 represented by circles. There are three data objects A, B and C. A triangle represents a read copy of a data object and a square the write copy. The ellipses represent the assignment of data to processors. For example, P_2 has read copies of A, B and C and the write copy of B. This figure also contains the *Data Object Catalog* at P_1 and the *Data Access Catalog* and *Shipping Catalog* at P_3.

based on this cost and the price it charges for e_{13}. We use very simple price setting algorithms for our experiments in section 3. They are described below.

A processor uses a *data access catalog* to inform local transactions of the accesses it supplies. The catalog contains the price and quality for each service provided by the processor. Figure 35 on page 127 shows the catalog maintained by P_3 . An entry in this catalog contains: 1) An order number, 2) The object ID, 3) Mode (read/write), 4) Price and 5) Quality. In the example, the quality is defined as -1.0 times the distance to the copy accessed. The quality of entry 7 (*Write*(*B*)) is −2.0. This catalog also shows that there can be multiple entries with different prices and qualities for a given access.

Processor P_3 also maintains a *shipping catalog* that lists the prices charged to ship data to other nodes. Figure 35 on page 127 shows the shipping catalog at P_3. The price per message to ship data to P_2 is \$0.25. This includes P_3's price for link e_{31} and P_1's price for e_{12}. The processors set their link prices based on demand. The pricing algorithm for our simulation study is presented in section 3. The same subcontract protocol exists for shipping data as for remotely processing data access operations.

A processor leases a copy of a data object only if it is profitable to do so. To make leasing business decisions, a processor records all data accesses it sells and all subcontracts it processes. In the example, P_1 records all of its sales of read and write access on object A . It also records all of the *Read*(C) and *Write*(C, *value*) subcontracts it sends to P_3, and all of the *Read*(B) and *Write*(B, *value*) subcontracts it sends to P_2. Processor P_1 measures its average revenue for read and write access to A and maintains its leases on these objects only if they are good investments. There is a system wide *interest rate* I that formalizes the definition of a good investment. If L_A^R is the read lease price of A, and R_A is the average *Read*(A) revenue per lease period, P_1 retains its read lease on A only if

$$R_A - L_A^R \geq I \bullet L_A^R.$$

$R_A - L_A^R$ is P_1's profit on *Read*(A) per lease period. $I \bullet L_A^R$ is the return on an investment of L_A^R dollars if the money were deposited in a "savings" account.

A processor acquires a new lease if it estimates doing so will be a good investment. Assume that P_1 pays P_2 and average of R_B dollars per lease period for subcontracting *Read*(B). Two lease prices are associated with a read copy of B : 1) The acquisition price L_B^A, and 2) The renewal price L_B^R. If P_1 acquires a read lease, it pays L_B^A for the first lease period, and L_B^R for subsequent renewals. (How these prices are set and distributed is described below).

P_1 acquires a read copy of B if

$$R_B - L_B^R \geq I \bullet R_B^A.$$

$R_B - L_B^R$ is the expected profit per lease period, and R_B^A is the initial investment needed to make this profit.

5.2.2 Data Object Agents

There are two types of agents associated with data objects. Each object A is controlled by a *manager M_A*, which resides with the write copy. The second type of agents are *local agents*. There is a local agent at each processor P_i and this agent acts as a representative to P_i for the data object managers located at other processors. In the example, M_C is located at P_3 and it uses the local agents at P_1 and P_2 to interact with these processors.

The data object manager is responsible for updating the read copies of the object. Assume that P_3 services a *Write(C, value)*. P_3 passes the update to M_C. The agent M_C sends the update to the local agent at P_2. M_C uses the shipping catalog and must pay P_3 \$0.25 to send the data to P_2. When the update arrives at P_2, the local agent submits the update operation at P_2. Since this update does not effect any other copies, the price charged by P_2 is the local read price for C. This is \$0.50 in the example. After the update completes, the local agent at P_2 sends a response and a *bill* to M_C. When this message arrives, M_C informs P_3 that the update is complete and then sends a *check* for \$0.50 to the local agent at P_2.

The data object manager sets the lease renewal and acquisition prices for the object. The goal is to set the prices that maximize the object's profit. The renewal price for the read copy is set based on the demand for read copies. Many price functions are possible, and we propose a simple algorithm in section 3. The read acquisition price is simply the renewal price plus the cost of shipping a copy to the processor. In the example, the read renewal price of C is \$20 and all objects require 1000 messages to ship a copy. So, the read acquisition price at P_1 is \$220.

Since there is only one write copy, the data object manager uses an auction to set the write renewal price. The write renewal price has a lower bound which must cover the cost of updating the read copies. This lower bound is determined by averaging the update cost per copy over time. Any bid below this value is rejected.

The data object manager broadcasts all changes in renewal and acquisition prices to the local agents. Each local agent posts these prices in a *data object catalog*. Figure 35 on page 127 contains the data object catalog at P_1. An entry in this catalog has the following fields: 1) The object ID, 2) The mode (R/W), 3) The acquisition price, 4) The renewal price and 5) An order number.

A processor P_i uses the local data object catalog to determine renewal and acquisition prices. To order an object, P_i sends a request to the local agent and pays the fee listed in the catalog. This request and the fee are forwarded to the data object manager. When receiving a lease request, the manager pays the local processor P_j to ship a copy of itself to the processor ordering the copy. When the copy arrives, P_i can start using it to supply service to transactions and other processors. The local agent at P_i also collects and forwards renewal fees when P_i renews a lease, and informs the manager when a lease is not renewed.

5.2.3 Transactions

A transaction T enters the system with an initial endowment of money M_T and sequentially submits read and write operations. T uses the data access catalog to order these services from the local processor. In the example, if T arrives at P_3 and wants to issue *Read*(A), T can either order catalog entry 1 or entry 2. The decision is based on the amount of money the transaction possesses. In our model, the transaction purchases the best service that it can afford. If T cannot afford any service, for example it has only $0.50 remaining, it aborts. Other policies such as queueing are possible and are left for future work.

5.3 Performance Evaluation

We implemented and simulated the data management economy to measure its performance. The data management problem studied in this chapter is not mathematically tractable without relying on simplistic analytical models of the underlying distributed system.

The economy was simulated on the 5 processor distributed system depicted in Figure 36 on page 131. There were 10 data objects, and each object contained 1000 records. Individual records were the granularity of transaction access. The processor service time for an access on local data was 1 time unit. The processors serviced the local data accesses sequentially. The transmission delay per message was 2 time units for all links. Each access request required 1 message, and shipping an entire data object required 1000 messages.

The processors' pricing policy for local resources was static with the following prices:

1. Read access : $0.50

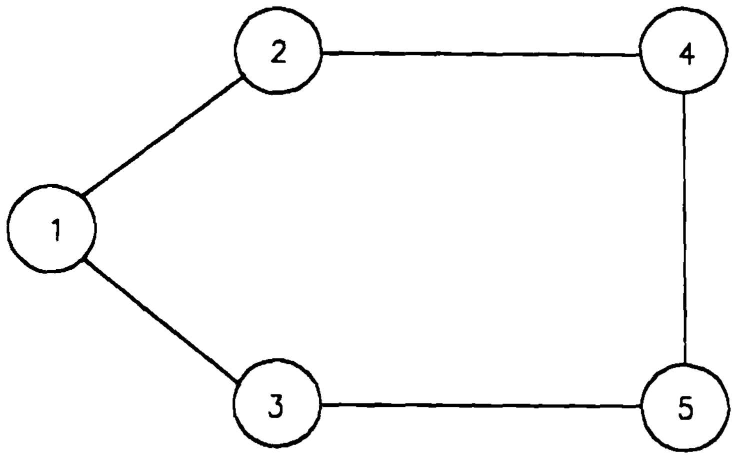


Figure 36. Data Management Distributed System

2. Write access : \$0.50

3. Link use : \$0.01 per message

Each processor P_i marked up all of its costs by 10%. In the example, P_3 charges P_1 \$2.20 for *Write*(C, *value*). P_1 would charge local transactions \$2.47 = (1.1)(\$2.20) + \$0.05 for *Write*(C, *value*).

In the simulations, a transaction T entered the system with $M_T = \$10$ and randomly submitted between 1 and 10 access operations. No transactions aborted due to insufficient funds. Applying simple algebra to to Figure 36 on page 131 shows that the maximum cost for 10 data accesses is less than \$10.

Each data object manager used an English Auction to sell the write copy. The managers' algorithm for setting read renewal prices was to charge $2 times the number of leases held.

We compared the Data Management Economy to two other solutions to the data management problem. The first is the *single copy* (SC) approach. In this scheme, there is a single read/write copy of each object D_j. All read and write access operations are sent to the processor P_i that contains the copy. The copies are uniformly distributed over all processors, and in the simulation each processor had two local data objects. The second policy is the *read everywhere* (RE) approach. Every processor P_i has a read copy of every data object D_j. This means that all read accesses can be serviced locally. There is a single write copy of each data object D_j and the write copies are uniformly assigned to the processors (each processor has two write copies). All write accesses on object D_j are sent to the P_i that has the write copy. This processor updates all read copies and sends a response to the requesting processor.

5.3.1 Ratio of Reads to Writes

To solve the data management problem, the economy must determine how many read copies of each object should exist. The optimal choice for the number of copies of object D_j, denoted $c(j)$, depends on the system wide ratio of read access to write access on this object. If read access is dominant, many copies should be available. This ensures that most transactions have their $Read(D_j)$ serviced locally or at a near by processor. Each $Write(D_j, value)$ requires that all $c(j)$ copies be updated, and the time required to process a write increases with the number of copies. So, if $Write(D_j, value)$ is dominant there should be few read copies.

Figure 37 on page 133 shows the performance of the data management economy, the single copy policy and the read everywhere policy. The Y-axis in this figure represents mean transaction response time. The X-axis represents the *write probability* P_W. At $P_W = x$, a transaction T that enters the system and accesses data object D_j performs a write with probability $P_W(D_j) = x$. In these experiments, $P_W(D_j)$ was the same for all objects.

The performance of the single copy policy is uniform for all values of x. If there is only a single copy of D_j, $Write(D_j, value)$ does not generate any read copy updates. A $Write(D_j, value)$ requires exactly

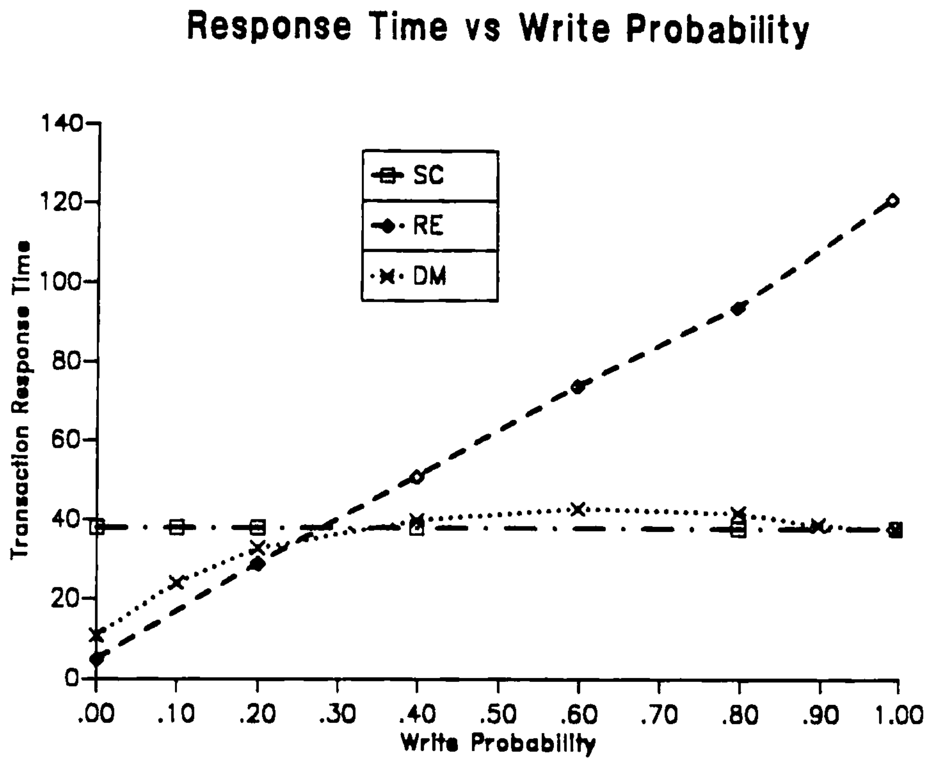


Figure 37. Adapting to Read/Write Ratio: This figure shows the mean response times of the data management economy (DM), the single copy system (SC) and the read everywhere system (RE).

as much processing as a *Read*(D_i) and the system is not effected by the read/write mix. The single copy policy achieves the best performance for $P_W \geq 0.40$.

The read everywhere policy exhibits the best response time for $P_W \leq 0.30$. In these experiments, the mean response time of the read everywhere policy increases linearly with the write probability. Under this policy, each *Write*(D_i, *value*) requires significantly more work than a *Read*(D_i). Assume that the system uses the read everywhere policy. Let t_W be the average response time for a write access (including updating all copies) and let t_R be the average read response time. Also, assume that these times are constant for all P_w. At write probability $P_W = x$, the expected response time R of a data access is

$$E[R \mid (P_W = x)] = x \cdot t_W + (1 - x)t_R$$

which simplifies to

$$E[R \mid (P_W = x)] = x(t_W - t_R) + t_R.$$

This means that the average access response time is a linearly increasing function of x. The same holds for the mean transaction response time, which agrees with Figure 37.

The data management economy exhibits near optimal response times for all values of P_W. Its worst performance is at $P_W = 0$. At this value, the response time is approximately double the read every policy. The reason for this is described below. At all other values of P_W, the data management economy is no more than 13% worse than minimum response time of the single copy and read everywhere policies.

For simple models, it is possible to show that for any value of $P_W = x$, the optimal policy is either read everywhere or single copy. This means that for $0.1 \leq P_W \leq 1.0$, the data management economy is within 13% of the minimum response time over all algorithms. As an example, assume that for a given $P_W = x$, the average read and write access response times are linear functions of the number of copies of each data object, denoted C. That is,

1. $t_R(C) = a_R C + b_R$.
2. $t_W(C) = a_W C + b_W$.

The average response time at $P_W = x$ is

$$R(C) = (1 - x)t_R(C) + x t_W(C).$$

To find the optimal C, take the derivative of $R(C)$. This is

$$R'(C) = (1 - x)a_R + x a_W.$$

The derivative is a constant, which means that $R(C)$ takes its minimum value at either $C = 0$ or $C = N$, where N is the number of processors. This analysis holds for all x, and under these simple assumptions either the read everywhere or single copy policy is optimal for all x.

Two economic factors cause the data management economy to adapt the number of read copies of each object D_j to the read/write ratio. These are:

1. The total revenue that all processors earn by selling $Read(D_j)$ decreases as P_W increases.
2. The read lease price for D_j increases linearly with the number of copies $c(j)$.

For a given system wide arrival rate, this means that as P_W increases, the processors as a whole can afford fewer read copies.

Figure 38 on page 136 shows how the data management economy adapts the number of copies of each object D_j to read/write ratio for it. In this example, there is a single data object A. The read lease price is defined to be 5/3 times the number of leases held. Assume that on average, ten transaction accessing A arrive at each processor during a lease period. Also assume that the processors charge \$0.50 for a $Read(A)$. If $P_W = 0$, each processor can earn \$5 selling $Read(A)$. The system wide revenue is \$15. The lease renewal price if three copies are leased is $\frac{\$5}{3} \times 3 = \$5$, and 3 copies can be afforded by the system as a whole. (To avoid complicating the example, the interest rate $I = 0$). At $P_W = 0.50$, half of the accesses are writes and only \$7.50 worth of $Read(A)$ business enters the system per lease period. The total lease price for 3 leases is \$15, and the total for two leases is \$6.67. This means that the system as a whole can only afford two copies. In the example of Figure 38 on page 136, P_1 subcontracts its $Read(A)$ to P_2 and P_3. Finally, at $P_W = 0.80$, the total $Read(A)$ revenue in the economy is \$3 and only one copy is affordable. How the copies are assigned to the processors is described in a subsequent subsection.

The data management economy uses decentralized decision making to compute the number of read copies of each object. The business strategies of the processors are decoupled, and P_i uses only local information to estimate its revenue. The system as a whole is not completely decentralized, however. The lease prices for each object are computed at a central headquarters and are globally available to all processors. The lease requests, renewals and cancellations are also processed at a central headquarters. Cancellations, requests and renewals, and the lease price changes they cause do not occur frequently. In the simulation study, the lease period was 1000 time units. The overhead of the centralized decision making and global price distribution can be controlled by setting the length of the lease period.

In the experiments of this section, the data management economy adapts itself to any read/write ratio without external intervention. The economy is not *completely* self tuning, however. There is a subtle interaction between the following factors:

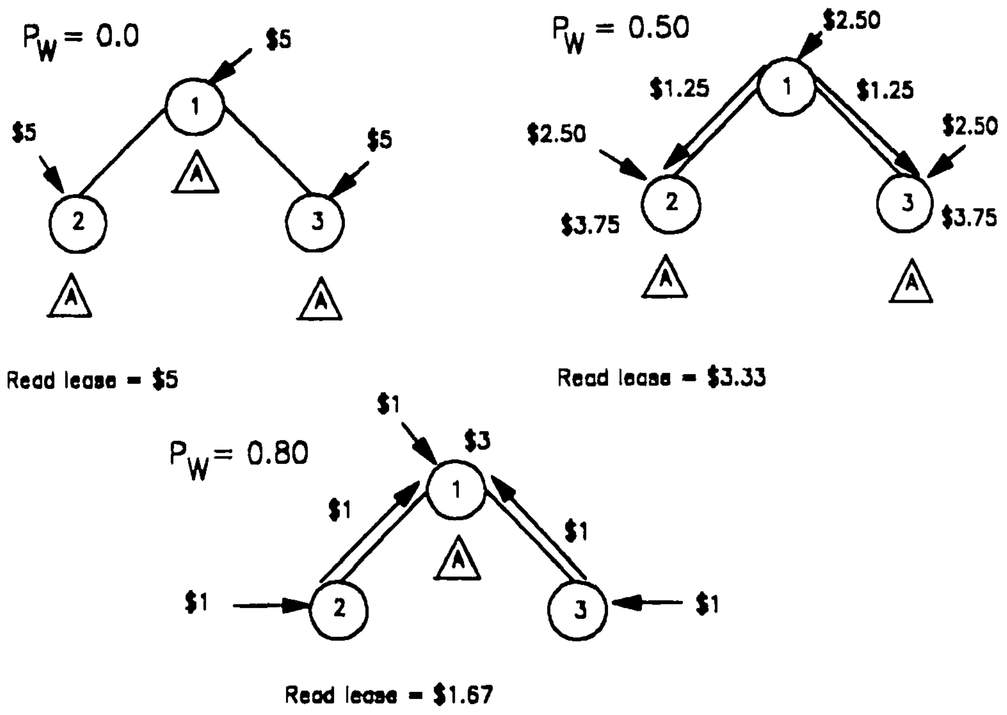


Figure 38. Adapting to Read/Write Ratio: This figure shows how the data management economy adapts to the read/write ratio. There are three processors represented by circles, and one data object A. On the average 10 transactions accessing A arrive at each processor during a lease interval. The read access price is $0.50 at all processors. The read lease price is set at 5/3 times the number of copies.

1. The lease price function
2. The transaction arrival rates
3. The wealth of the transactions
4. The price charged by the processors for read/write access

As an example, assume that in Figure 38 on page 136 the processors charge $3.00 for each read access. At $P_W = 0.8$, each processor can earn $10 \times 0.20 \times \$3 = \$6$ selling *Read*(A), provided that the transactions can afford this price. This means that each processor can afford a read lease and there will be three copies of A as opposed to the 1 copy in the previous example. This phenomenon also explains why the data management economy is worse than the read everywhere policy at $P_W = 0$

in the experiments of Figure 37 on page 133. In this system, even at $P_W = 0$, the total revenue earned selling $Read(D_j)$ is less than the total cost cost of 5 read leases. So, the system cannot afford to place a read copy at each processor.

Automating the subtle interplay between the transaction arrival rates, object access probability, P_W, transaction endowment, processor price algorithms, and lease price algorithms is an important area for future research. For the ideas presented in this dissertation to achieve their full potential, these issues must be addressed.

5.3.2 Adapting to Entropy

In the experiments in the previous section, the write probability $P_W(D_j)$ was the same for all data objects. If $P_W(D_j)$ is considered as write "heat" or energy, the system was in a state of total entropy. A policy or algorithm that examines the individual characteristics of the data objects has nothing to exploit for improving performance compared to policies that look at P_W for the entire system.

Figure 39 on page 138 demonstrates the data management economy's ability to improve performance by adapting to the read/write characteristics of individual objects. In this figure, the Y-axis represents mean transaction response time. Two sets of experiments are depicted in this figure. They represent systems with average write probability $P_W = 0.5$ (data management economy DM:P = 5, read everywhere RE:P = 5) and $P_W = 0.2$ (DM:P = 2, RE:P = 2). The single copy policy has the same performance in both systems. In the experiments of the previous section, the read everywhere policy is the best when $P_W = 0.2$ and the single copy policy is best when $P_W = 0.5$.

The X-axis in Figure 39 on page 138 represents the read/write entropy in the system. At point x, half of the objects have $P_W(D_j) = P_W + x$ and half have $P_W(D_j) = P_W - x$.

The performance of the single copy policy is not effected by P_W or x. This is because when there is a single copy, the performance costs of reads and writes are the same. The performance of the read everywhere policy is substantially better at $P_W = 0.2$ than at $P_W = 0.5$, as in the previous section. The read everywhere policy's performance is not effected by the value of x. The decreased update overhead of an object with $P_W(D_j) = P_W - x$ is offset by the increased cost of a copy with $P_W(D_k) = P_W + x$. The total update cost remains the same.

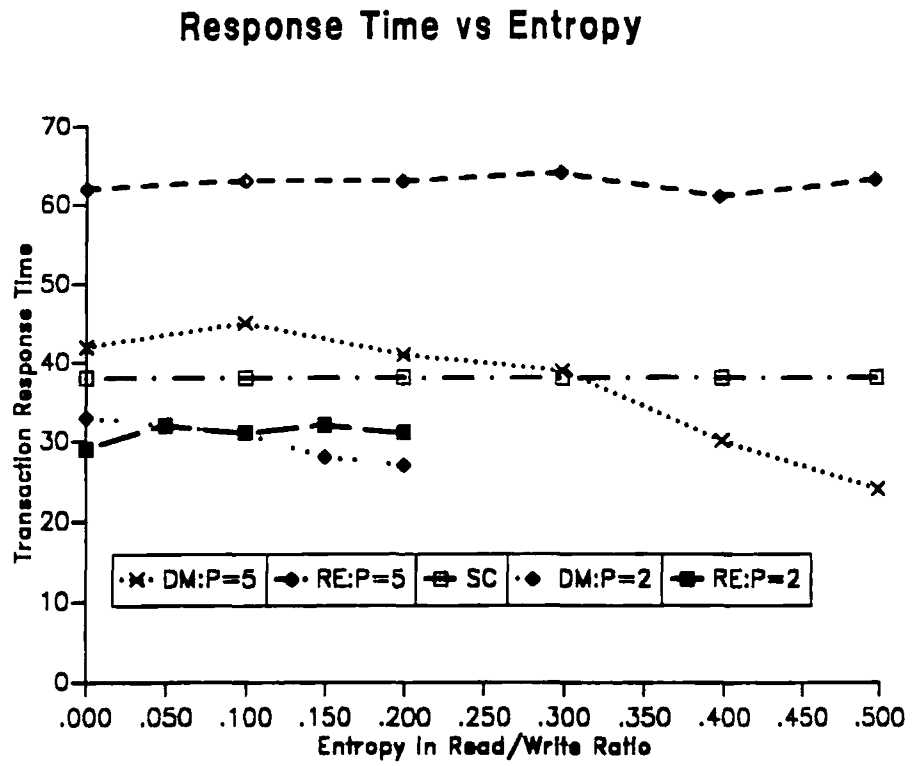


Figure 39. Performance vs Entropy: This figure plots the performance of the system for write probabilities $P_w = 0.5$ and $P_w = 0.2$. The X-axis represents the variance in the individual write probabilities. At point x, half of the objects have $P_w(D_i) = P_w + x$ and half have $P_w(D_i) = P_w - x$. The five curves represent the data management economy and the read everywhere policy with $P_w = 0.2$ (DM:P = 2, RE:P = 2) and $P_w = 0.5$ (DM:P = 5, RE:P = 5). The single copy policy (SC) has the same performance in both systems.

The data management economy can use the individual characteristics of the data objects to improve performance. In the case of $P_w = 0.5$, the economy's performance is 10% worse than the single copy policy's when $x \leq 0.1$. For $x \geq 0.2$, there is enough diversity (energy) to allow the economy to improve performance and be as effective as the single copy policy. For $x > 0.3$, the economy is the best policy. The response time decreases by 40% percent at the extreme case of $x = 0.5$, and the economy is 30% better than the single copy policy. At $P_w = 0.2$, the read everywhere policy is the best at $x = 0$. For $x > 0.1$, the data management economy is the best policy.

The experiments in this section demonstrate that the data management economy can exploit diversity in the read/write probabilities of the objects to improve performance relative to policies that are based on global averages. No manual intervention is required for the economy to adapt to the the diversity in the write probabilities.

5.3.3 Adapting to Locality

A second source of diversity that can be exploited is *locality* in the transactions' reference patterns. In the previous experiments, a transaction T arriving a node P_i accessed all of the data objects with equal probability. Let $p(i,j)$ denote the probability of a transaction at P_i accessing object D_j. There are localities in the reference patterns if $p(i,j)$ is not constant for all i and j. For example, if $p(i,j) > p(k,j)$ for all other processors P_k, a copy of D_j should be assigned to P_i.

Figure 40 on page 140 depicts the results of experiments to evaluate the economy's ability to adapt to localities in the reference patterns. The Y-axis in this figure represents mean transaction response time. The X-axis represents the locality in the transactions' reference patterns. At point x, the transactions at a processor P_i exclusively access $x\%$ of the data objects. The subsets of data objects accessed at each processor were chosen randomly. For all values of x, the mean transaction length was 5 accesses. In this experiment, the write probability was $P_W = 0.4$ for all objects.

The performance of the read everywhere and single copy policies do not change as a function of locality of reference. Under the read everywhere policy, the response time of K reads on K distinct object is the same as reading one object K times. The same is true for writes. A similar argument holds for the single copy policy.

The data management economy's performance improves as locality increases. For the range $100 \geq x \geq 60$, the response time changes by less than 10%. In the range $60 \geq x \geq 20$, the mean transaction response time decreases linearly with increasing locality. At $x = 20$, the economy is 30% better than the single copy policy.

Figure 41 on page 141 demonstrates how the data management economy adapts the assignment of copies to processors based on referential locality. In this example, transactions at P_1 and P_2 only update object C. Transactions at P_3 only read B. Read and write access prices are \$10 and the ship price per link is \$2.

Initially, P_2 is the only processor with a read copy of B. P_3 subcontracts *Read*(B) to P_1, which in turn subcontracts to P_2. Assume that 10 transactions per lease period arrive at P_3. P_2 earns \$100 per lease

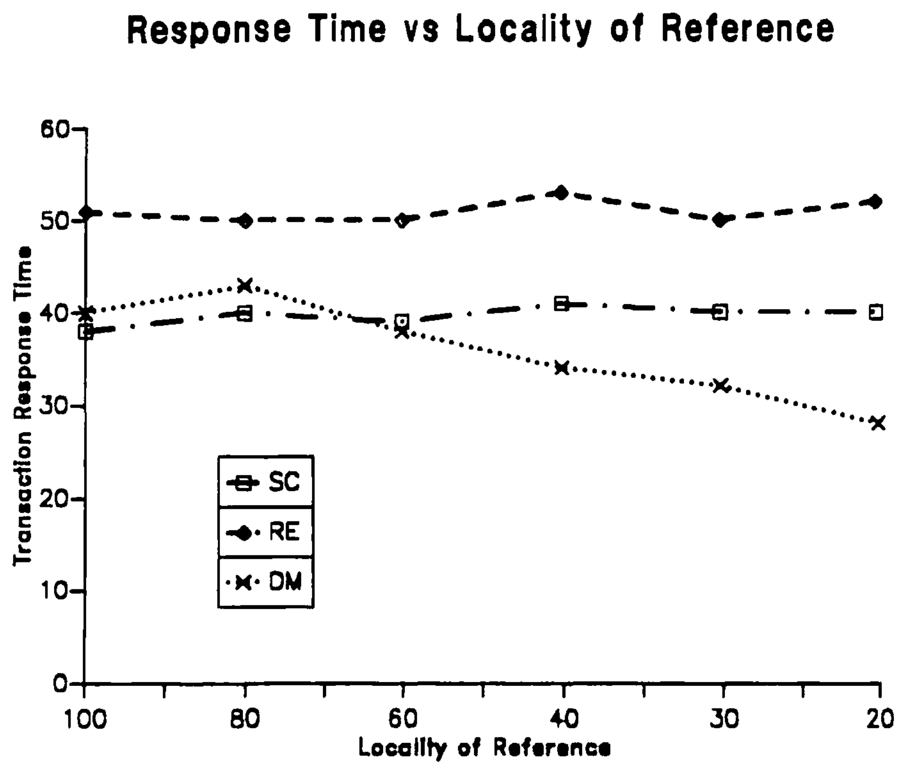


Figure 40. Performance vs Locality of Reference: This figure plots the performance of the single copy policy (SC), the read everywhere policy (RE) and the data management economy (DM) as a function of the locality of reference. The Y-axis represents average transaction response time. The X-axis represents the percent of the total number of data objects accessed by transactions at a given processor.

period selling *Read*(B), and P_1 loses \$110 and P_3 loses \$130. P_1 and P_3 will each order a read lease for B (provided that the cost is less than \$100). When this happens, P_3 no longer subcontracts its *Read*(B) business, and P_1 and P_2 have their *Read*(B) revenue fall to 0 and will not renew the read leases on B.

In the example, the only write copy of C is at node P_3. P_2 subcontracts its *Write*(C) to P_1 which in turn subcontracts to P_3. Assume that the arrival rate at processors P_1 and P_2 is 10 transactions per lease period. P_3 earns \$200 per lease period, and P_1 and P_2 lose revenue of \$220 and \$160 respectively. If the interest rate $I = 0$, the processors are willing to bid the following for the write copy of C: 1) 220, 2) 160, 3) 200. So, P_1 wins the next auction for the write lease of C. At this point, P_1 earns \$200 per lease period on *Write*(C) while P_2 loses only \$120 by not holding the lease. The write

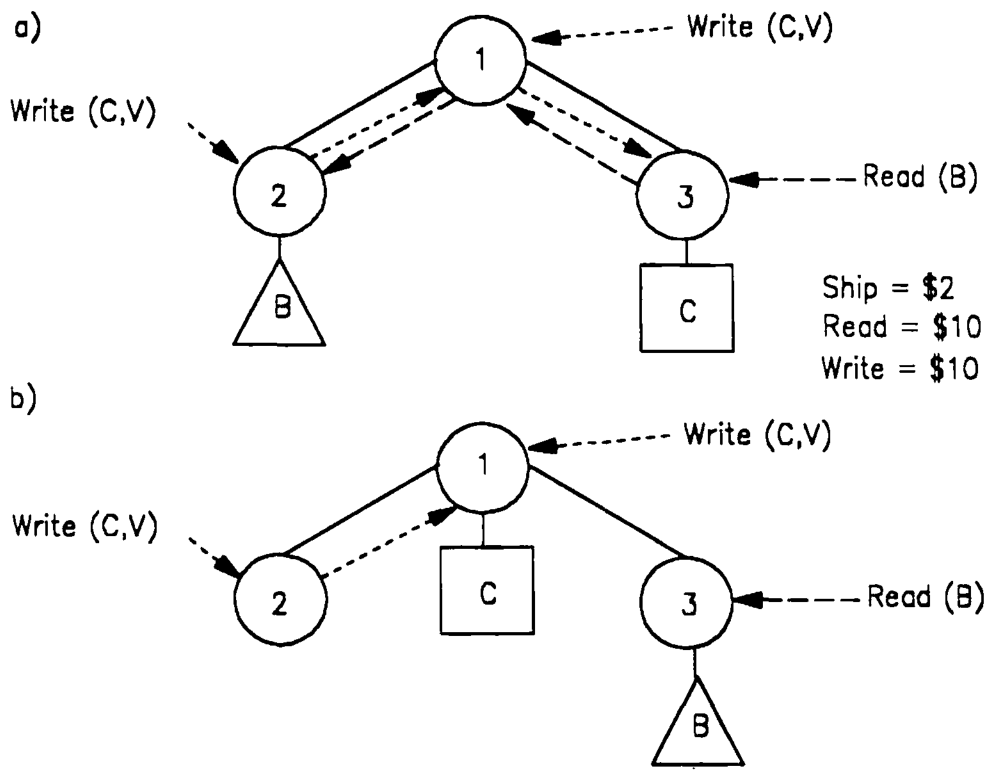


Figure 41. Adapting to Locality of Reference: This figure shows how the data management economy adapts to locality in the reference patterns. Transactions at P_1 and P_2 exclusively update object C. Transactions at P_3 only read B.

copy will not migrate to P_2 and the system stabilizes. The new configuration is depicted in part b of Figure 41 on page 141.

5.3.4 Lease Prices

In the data management economy, the price for a resource should reflect the underlying demand for the resource. Agents make their decisions based on resource prices, and having the prices accurately reflect resource demands can increase the economy's effectiveness. Resource prices can also help system management and capacity planning by identifying bottleneck and surplus resources.

In the experiments of this section, the read, write and ship prices charged by the processors are constant. Even with this simple price policy, the economy dramatically improves performance. To

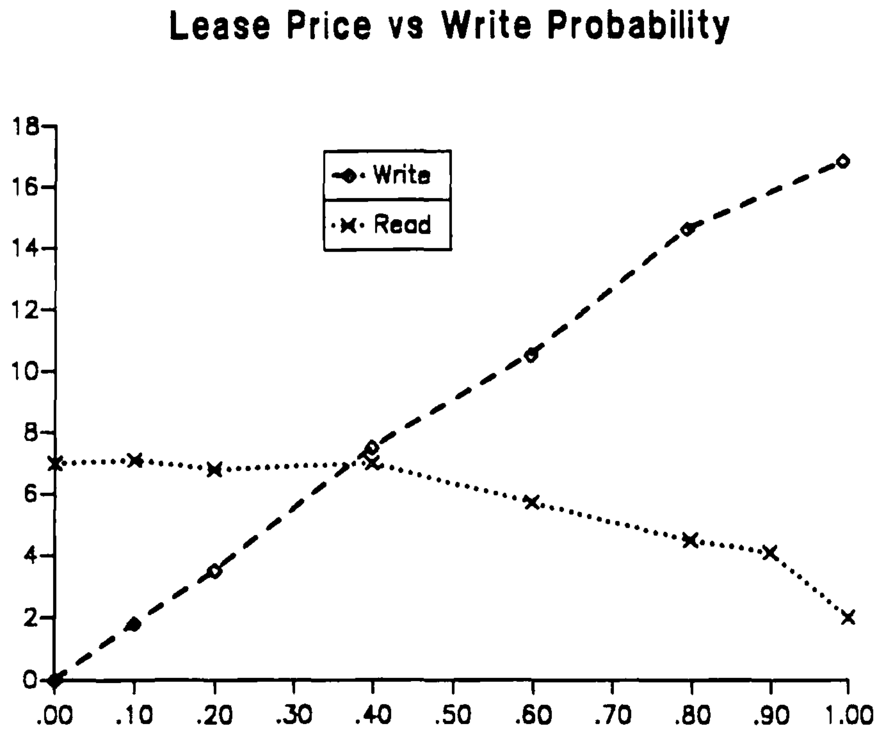


Figure 42. Lease Prices vs Write Probability: This figure plots the average lease price versus the probability P_w of an object access being a write. The two curves represent the average read lease price and the average write lease price over all objects.

evaluate the relationship between resource demands and prices, we ran experiments to measure the effect of the system wide write probability on read and write lease prices.

Figure 42 on page 142 depicts the results of these experiments. The X-axis represents the system wide write probability P_w. At point x, a data object D_i that is accessed by transaction T is updated with probability P_w. In this experiment, the mean number of objects accessed by a transaction is 5. The Y-axis in this figure represents average lease price. Each time a processor renews, cancels or requests a lease on a data object, the price is recorded. The two curves in this figure represent the average read lease price and write lease price over all objects.

The average write lease price is a linearly increasing function of P_w. This accurately reflects the transactions' demand for write access to objects, which increases linearly with P_w. Each data object manager uses a distributed English auction to allocate (lease) the write copy to a processor. In the data management economy, the auction model sets resource prices that accurately reflect the underlying demand for the resources. This contrasts with the load balancing economy in which examples showed that the auction model did not set prices that accurately reflect demand. In the load balancing economy's auctions, the resource price is set by the wealthiest job and the maximal wealth is not directly related to the number of jobs bidding. In the data management economy, the number of potential bidders for a write lease is fixed. As P_w increases, the wealth of each bidder increases linearly. In the examples of Figure 22 on page 69 in the load balancing economy, if the number of jobs remains the same and the wealth of each job increases, the sale price will also increases. This explains the success of auctions in the data management economy.

Figure 42 shows that the read lease prices do not accurately reflect the demand for read access to data. The read demand decreases linearly with P_w but the read lease price is nearly constant in the interval $0 \leq P_w \leq 0.4$, and decreases nearly linearly in the interval [0.4, 0.9].

The algorithm for setting the read lease price was to charge $2 times the number of copies leased. The write copy is also a read copy, which means that the minimum price for a pure read copy was $2. Since the price is a function of the number of copies held, this figure shows that too few read copies are demand when $P_w \leq 0.2$ and too many are held when $0.5 \leq P_w \leq 0.9$. This explains why the economy's performance is not as good as the read everywhere policy when $0 \leq P_w \leq 0.2$ and is not as good as single copy with $0.5 \leq P_w \leq 0.9$. It may be possible to improve the economy's performance by choosing another constant for the read lease price algorithm.

The demand for read copies is not only a function of P_w. The interest rate, shipping costs and topology also effect the demand for read leases. The same is true for write lease prices. However, there is a single write copy and the write demand aggregates over all nodes. There can be multiple read copies and the demand is per processor as opposed to system wide. So, the read lease demand is more subject to local considerations. Modifying the read lease price policy to account for these problems is an area for future work that may improve the economy's' performance.

5.4 Comparison with Related Work

In the performance experiments discussed in this chapter, the data management economy solves the *File Allocation Problem*. We compare the data management economy with previous work on the file allocation problem in this section. More general and extensive surveys can be found in [33, 88].

Most previous solutions to the file allocation problem have been based on minimizing a cost function. This approach was first used by Chu [18], and the most comprehensive cost function and constraints are also due to Chu [19]. Under these models, there are M files (data objects) and N processor nodes. The algorithm computes $N \bullet M$ binary decision variables X_{ij}. $X_{ij} = 1$ if a copy of D_i is assigned to processor P_j, and is 0 otherwise. The goal is to compute the X_{ij} that minimize a cost function. As an example, the cost function could be total communication cost. Let λ_{ij}^{R} be the rate at which read accesses for D_i are generated at P_j, and let λ_{ij}^{W} be the same for write accesses. If C_{jk} is the "cost" (e.g. - delay) of sending a message between P_j and P_k, the cost function to minimize is

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{N} 2(\lambda_{ij}^{W})X_{ik}C_{jk} + \sum_{j=1}^{N}\sum_{i=1}^{M}\lambda_{ij}^{R} \bullet (\mathrm{Min}\ \{C_{jk} \mid X_{ik} = 1\}).$$

The first term presents the total write cost, and the second term is the total read cost (Every read is routed to the "cheapest" copy).

In general, the cost minimization problem is subject to several constraints. One constraint is that there be at least 1 copy of each object, i.e. $\sum_{j=1}^{N} X_{ij} \geq 1$ for all i. Other constraints can include disk, CPU or communication capacity. The most general formulation of the cost function and constraints is due to Chu [19].

A main disadvantage of the cost minimization problem is that it is a 0-1 integer programming problem and is NP-Hard [28, 77]. Heuristic algorithms have been proposed and work well in simple configurations [14, 77]. An alternate approach has been to allocate the files singly. That is, solve M independent problems each with one file [14, 15, 28, 62, 73]. Even in this simple case, the problem can be NP-Hard. Furthermore, there is obviously interactions between the assignments of the individual files, which is ignored by this model.

The cost minimization problem suffers from other disadvantages. First, the models are too simplistic. For example, C_{jk} is assumed to be constant and not a function of the traffic between P_j and P_k. Secondly, no methodology for defining the costs is given. It is not obvious how the cost for disk space relates to the communication cost. This previous work also requires complete, centralized information and uses a centralized algorithm to compute the assignment. To minimize the function above, λ_{ij}^{R} and λ_{ij}^{W} most be known for all processors. The algorithms also compute static assignments, and must be re-executed if the inputs change.

There are two advantages to this set of algorithms. If the cost functions can be accurately defined, it is possible to solve the constrained optimization for some realistically large systems. In this case, the assignment is provably optimal [88]. There is also evidence that heuristic algorithms do provide near optimal assignments [33, 88].

The main advantage of the data management economy relative to the cost minimizations algorithms is that the economy uses decentralized computation, and simple statistical measurement of the parameters. Each processor independently measures λ_{ij}^{R} and λ_{ij}^{W}. The only necessary global information are the lease prices. This information is relatively small and changes slowly. An additional advantage of our work is that our simulation studies were based on realistic system models as opposed to the work above, which uses simple analytic cost functions. Finally, the data management economy is adaptive and does not compute static solutions.

The main disadvantage of the data management economy is that currently there is no methodology for choosing pricing functions. This is especially true for read/write access, link costs and read lease prices. Our simulation studies indicate that very simple policies work well, however. A more thorough study of pricing algorithms is an area for future work. A second disadvantage of the results of this chapter is that the economy does not compute provably optimal assignments. Attempting to use mathematical economics to prove optimality of the economy is also an area for further work.

The second category of solutions to the file allocation problem is based on analysis of queueing networks [33, 88]. A global performance metric, which is usually throughput or response time, is defined and an assignment of files to processors that optimizes the metric is computed. This work is similar to the material above in the sense that a global metric is defined and optimized. The main difference is that the queueing delays at CPU and communication resources are explicitly computed instead of relying on statically defined costs.

Under this model, the file allocation problem is solved in two stages. In the first stage, a performance function $F_i(x)$ is defined for each processor P_i. For example, $F_i(x)$ could be the mean response time of a file access at P_i when the arrival rate at P_i is x. The first stage algorithm computes a branching probability p_i for each node P_i. If Λ is the system wide arrival rate of file accesses (both read and write), $p_i \cdot \Lambda$ of the requests are routed to P_i. The first stage algorithm computes the branching probabilities $p_1^* p_2^*, \dots, p_N^*$ that minimize

$$\sum_{i=1}^{N} F_i(p_i \cdot \Lambda),$$

subject to

1. $\sum_{i=1}^{N} p_i = 1$
2. $0 \leq p_i \leq 1$.

The stage 1 optimization computes the processor loads that optimize the performance metric.

The second stage is to compute an assignment of files to processors that yields the desired branching probabilities. It is typically assumed that the files are infinitely divisible [33], and $0 \leq X_{ij} \leq 1$ instead of $X_{ij} \in \{0, 1\}$. Given the branching probabilities and the distribution of access over the files computing the X_{ij} is trivial.

For constrained topologies and under simplifying assumptions (exponential service time, poisson arrival process, etc.), several algorithms have been studied that compute the optimal $p_1^*, p_2^*, \dots, p_N^*$ [4, 13, 17]. This work assumes that the distribution of accesses over all files is known, and that the files are infinitely divisible. This means that the second stage computation is trivial.

If $X_{ij} \in \{0, 1\}$ is required, computing the assignment in the second stage is NP-Hard [33]. Heuristic search, linear programming and bin packing algorithms have been used to compute the X_{ij} in this instance [31, 32].

The work above is only applicable in restricted topologies, and some assume that all references are generated at a single node. Dowdy has extended throughput maximization to general topologies, but optimality is no longer guaranteed [33]. This algorithm also requires solving an integer programming problem to compute the assignments. In their survey, Dowdy and Foster demonstrate that computing a file allocation that minimizes communication delay can be modeled as a traffic flow assignment problem [52].

There are several disadvantages to the to the queueing based approaches to file allocation. First, most work assumes that the data objects are infinitely divisible. If this is not the case, computing the file assignment that yields optimal branching probabilities is NP-Hard. Secondly, most algorithms can only be applied in restricted topologies (e.g. - single source of file references [76], star networks [13, 17, 32, 45]). Third, complete information on the parameters is required and the algorithm is centralized. Finally, the solutions are optimal only for static, steady state. The main advantage to these algorithms is that optimality can be proven given the assumptions.

As with cost based algorithms, the data management economy has the following benefits compared to the work above: 1) Decentralized computation, 2) More realistic system model, 3) Adaptive, non-static assignments. The main disadvantages are the same as before. 1) No methodology for pricing algorithms and 2) Not provably optimal assignments.

5.5 Conclusions

This chapter presented an economy that manages replicated data objects in a distributed transaction processing system. Simulation based performance experiments demonstrate that the economy substantially improves performance. The economy was compared to the policies of 1) Maintaining a single copy of each object, and 2) Maintaining a copy of each object at every processor. The economy substantially improves performance relative to these policies.

The economy improves performance in two ways. First, the number of copies of each object is set based on the ratio of read access to write access for the object. If writes are most common, few copies are maintained to avoid update overhead. If, however, read access is predominant multiple copies are maintained to ensure that reads are serviced locally or at nearby processors. The second improvement is derived through exploiting localities in transaction access patterns. Copies of objects are assigned to the processors at which they are most frequently accessed. We provided several examples that illustrate how the economy uses supply and demand to achieve the performance improvements.

The data management economy solves the File Allocation Problem. We presented a brief survey of previous work on this problem, and compared it with the economy. The main advantages of the data management economy relative to previous work are: 1) Decentralized decision making, 2) Realistic system models, 3) Computational tractability. The main disadvantage is that the economy's assignments are not provably optimal.

We studied the relationship between resource prices and demand. In this economy, an English auction model was shown to set lease prices for write copies that accurately reflect the demand for write access. Our algorithm for setting read lease prices was shown to be less successful. We are currently exploring alternative policies.

Several avenues for further work were presented. The interactions between the processors' and data object managers' pricing algorithms and the transaction endowments that should be addressed. We used simple pricing algorithms in our experiments. More sophisticated policies may be able to further improve performance. Finally, our economy is not currently compatible with concurrency control based on locking. Incorporating concurrency control is necessary to more realistically model transaction processing systems.

6.0 Summary and Directions for Future Work

This thesis studied resource sharing problems in distributed computer systems. We proposed that a distributed system should be structured as a society of competing microeconomic agents. We demonstrated the effectiveness and generality of this approach by applying our ideas to the diverse problems of *load balancing*, *flow control* in virtual circuit networks, and *data management*.

Our load balancing economy improved mean job waiting time compared to a representative, non-economic load balancing algorithm. We showed that our economy implements a broad spectrum of load balancing strategies. At one extreme, the economy has a high migration rate and migrates short jobs. At the other extreme, the economy migrates long jobs and has a low migration rate. We demonstrated that the economy's load balancing strategy can be tuned to relative speeds of CPU vs communication and that this gives better performance. The economy was shown to implement flow control by decreasing throughput to improve response time. We also evaluated the effectiveness of job wealth as a priority mechanism. Our experiments revealed that a simple learning mechanism for the jobs substantially improved performance and counter acted the effectiveness of wealth as a priority mechanism. Finally, we showed how the economy limits complexity by being inherently decentralized and modular.

The flow control economy demonstrated how economies effectively deal with diversity. Each user of a virtual circuit can independently set its individual throughput-delay goals. This improves on previous work which has either: 1) Defined a system wide throughput-delay goal, or 2) Has assumed all circuits have the same throughput-delay goals (e.g. - Maximize throughput). We proposed ***Pareto-optimality*** as a new definition for optimal resource allocations for the flow control problem. We presented a formalization of fairness criteria and compared Pareto-optimality with previous definitions of optimality and fairness. We extended tools provided by mathematical economics and proved that the resource allocations computed by our economy are Pareto-optimal. We also proved that a Pareto-optimal allocation exists for arbitrary network. Proving convergence in an economy is an extremely difficult problem, and we did not prove convergence for the flow control economy. An extensive simulation study demonstrated that the economy rapidly converged to an optimal allocation over a broad set of network parameters. We also discussed the integration of circuit routing with flow control. The flow control economy demonstrated the power of mathematical economics as a tool for modelling resource sharing problems in distributed computer systems. It also demonstrated that the models must be altered to realistically model the computer systems.

The data management economy solved a general version of the *file allocation problem.* This economy improved mean transaction response time compared to simple system wide policies. The improvements were achieved in two ways. First, the number of copies of a data object was chosen based on the ratio of write access to read access. Secondly, localities in transaction reference patterns were detected and exploited. The economy was tested on a more realistic system model that previous work. This economy demonstrated how economic competition resolves conflicting goals. There is a trade-off for the number of copies of an object in the system. Many copies decreases read access time, but increases write access time. A single copy eliminates update overhead, but increases read access time. The economy resolved these conflicts over a broad range of read/write ratios. The data management economy and the examples from this chapter highlighted the importance of the algorithms that set resource prices. Our experiments demonstrate the simple policies work well. Finally, the data management economy demonstrated that economic models can efficiently allocate *logical* resources (access to data).

This thesis opens up several opportunities for further work. This work falls into two categories. The first are extensions to the economies presented in this thesis. The load balancing economy experiments showed that the economy can have its load balancing strategy tuned for a specific ratio of CPU to communication power. This tuning requires manual intervention, however. Automating the tuning is a possible area for further work. We also discussed that the economy implements flow control. Currently there is no mechanism for setting the desired throughput-response goal. Providing such a facility is an area for future research. Two avenues for work are left open by the flow control economy. The first is proof of convergence of the tatonement process or a counter example. The second is integration of alternate models of VC throughput-delay goals. Our model is very general, but does not exhaust the set of possible goals. Finally, the data management economy has three opportunities for more research. One is determining if more complex price setting algorithms improve performance. Secondly, it was shown that there are interactions between the transactions' endowments, the processors' pricing algorithms and the data managers' pricing algorithms. Automatically resolving these interactions would enhance the usefulness of the economy. The final possible extension of the data management economy is including concurrency control in the economy.

The second avenue for further work is applying the microeconomic paradigm to other problems in distributed systems. Doing so will further demonstrate the generality of this approach. In this thesis we focused on the performance achieved by resource sharing algorithms. It may be possible to apply this approach to problems of reliability and error recovery. Finally, an extremely promising area for research is further relaxing the assumptions made by mathematical economics. This will allow modelling a broader set of distributed system problems. The two most important assumptions are: 1) Agents possess complete, apriori knowledge of their resource requirements. 2) The set of agents is fixed. Overcoming these assumptions is an interesting theoretical challenge, and would provide computer science with a powerful new set of tools.

Bibliography

[1] Kenneth J. Arrow and Michael D. Intrilligator, editor. *Handbook of Mathematical Economics.* North-Holland Publishing Co., Amsterdam, 1981.

[2] N. Abramson. The ALOHA System - Another Alternative for Computer Communication. *Proceedings AFIPS Fall Joint Computer Conference,* pages 281-285, AFIPS Press, Montvale, NJ, 1970.

[3] N. Abramson. The Throughput of Packet Broadcasting Channels. *IEEE Transactions on Communications*, COM-25(1), January 1977.

[4] S. R. Arora and A. Gallo. Optimal Sizing, Loading and Reloading in a Multi-level Memory Hierarchy SysteM. *Proceedings AFIPS 1971 Spring Joint Computer Conference*, 38, 1971.

[5] K. Arrow and F. Hahn. *General Competitive Analysis.* Holden Day Publisher, San Francisco, 1971.

[6] Amnon Barak and Amnon Shiloh. A Distributed Load Balancing Policy for a Multicomputer. *Software Practice and Experience*, 15(9):901-913, September 1985.

[7] Phillip A. Bernstein and Nathan Goodman. Concurrency Control in Distributed Database Systems. *Computing Surveys*, 13(2), June 1981.

[8] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[9] Dimitri Bertsekas and Robert Gallager. *Data Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[10] Kadaba Bharath-Kumar and Jeffrey Jaffe. A New Approach to Performance Oriented Flow Control. *IEEE Transactions on Communication*, COM-29(4), April 1981.

[11] Andreas Bovopoulos and Aurel Lazar. Decentralized Algorithms for Optimal Flow Control. *Proceedings of the 25th Annual Allerton Conference on Communication, Control and Computing*, September 1987.

[12] Raymond M. Bryant and Raphael A. Finkel. A Stable Distributed Scheduling Algorithm. *Proceedings of the Second International Conference on Distributed Computing Systems*, pages 314-232, April 1981.

[13] J. P. Buzen and P. Chen. Optimal Load Balancing in Memory Hierarchies. *Proceedings of IFIP*, North-Holland, Amsterdam, 1974.

[14] R. G. Casey.
Allocation of Copies of a File in an Information Network.
Proceedings AFIPS 1972 Spring Joint Computer Conference, AFIPS Presss, Arlington VA, 1972.

[15] K. M. Chandy and J. E. Hewes.
File Allocation in Distributed Systems.
Proceedings International Symposium on Computer Performance Modeling, Measurement and Evaluation, March 1976.

[16] D. Chazan and W. Miranker.
Chaotic Relaxation.
Linear Algebra and Its Applications, 2, 1969.

[17] P. Chen.
Optimal File Allocation in Multilevel Storage Systems.
Proceedings of AFIPS 1973 National Computer Conference, 42, 1973.

[18] Wesley W. Chu.
Optimal File Allocation in a Multiple Computer System.
IEEE Transactions on Computers, C-18(10), October 1969.

[19] Wesley W. Chu.
Optimal File Allocation in Computer Networks.
in N. Abramson and F.F. Kuo, editor, *Computer-Communication Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[20] E.F. Codd.
A Relational Data Model for Large Shared Data Banks.
CACM, 13(6), June 1970.

[21] C.J. Date.
An Introduction to Database Systems.
Addison-Wesley Publishing Co., Reading, Mass., 1982.

[22] Randall Davis and Reid G. Smith.
Negotiation as a Metaphor for Distributed Problem Solving.
Artificial Intelligence, 20:63-109, 1983.

[23] Harvey M. Deitel.
An Introduction to Operating Systems.
Addison-Wesley Publishing Co., Reading, MA, 1984.

[24] K. Eric Drexler and Mark S. Miller.
Incentive Engineering for Computational Resource Management.
in B. A. Huberman, editor, *The Ecology of Computation*, North-Holland, Amsterdam, 1988.

[25] Alex Dupuy.
Network Simulation Testbed (NEST) User Manual.
Computer Science Dept. , Columbia University CS-NEST, 1986.

[26] Derek L. Eager, D. Lazowska, and John Zahorjan.
Adaptive Load Sharing in Homogeneous Distributed Systems.
IEEE Transactions on Software Engineering, SE-12(5), May 1986.

[27] Derek L. Eager, Edward D. Lazowska, and John Zahorjan.
A Comparison of Receiver-Initiated and Sender-Initiated Dynamic Load Sharing.
Dept. of Computer Science University of Washington 85-04-01, April 1985.

[28] Kapali P. Eswaran.
Placement of Records in a File and File Allocation in a Computer Network.
Proceedings IFIP Conference, Stockholm, Sweden, August 1974.

[29] Donald F. Ferguson.
Dynamic Distributed Load Sharing Algorithms - A Survey.
Computer Science Dept., Columbia University, May 1985.

[30] Domenico Ferrari.
Computer Systems Performance Evaluation.
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

[31] D.V. Foster and J.C. Browne.
File Assignment in Memory Hierarchies.
Proceedings Modeling and Performance Evaluation of Computer Systems, North-Holland, October 1976.

[32] D.V. Foster, L.W. Dowdy, and J.E. Ames.
File Assignment in a Computer Network.
Computer Networks, 5:341-349, September 1981.

[33] Lawrence Dowdy and Derrell Foster.
Comparative Models of the File Assignment Problem.
Computing Surveys, 14(2), June 1982.

[34] C.J. Gao, J.W. Liu, and M Riley.
Load Balancing Algorithms in Homogeneous Distributed Systems.
Proceedings of the 13th International on Parallel Processing, 1984.

[35] C.J. Gao, J.W. Liu, and M Riley.
Load Balancing Algorithms in Homogeneous Distributed Systems.
University of Illinois at
Urbana-Champaign UIUC-DCS-84-1168, 1984.

[36] Lionel M. Ni and Ching-Wei Xu and Thomas B. Gendreau.
A Distributed Drafting Algorithm for Load Balancing.
IEEE Transactions on Software Engineering, SE-11(10), October 1985.

[37] Mario Gerla and Leonard Kleinrock.
Flow Control: A Comparative Survey.
IEEE Transactions on Communication, COM-28(4), April 1980.

[38] G.M. Heal.
Planning Without Prices.
Review of Economic Studies, 36:346-362, 1969.

[39] G.M. Heal.
The Theory of Economic Planning.
North-Holland Publishing Co., Amsterdam, 1973.

[40] W. Hildenbrand and A.P. Kirman.
Introduction to Equilibrium Analysis.
North-Holland Publishing Co., Amsterdam, 1976.

[41] Y.C. Ho, L. Servi, and R. Suri.
A Class of Center-Free Resource Allocation Algorithms.
Large Scale Systems, 1:51-62, 1980.

[42] Man-Tung Hsiao and Aurel Lazar.
A Game Theoretic Approach to Optimal Decentralized Flow Control of Markovian Queueing Networks with Multiple Controllers.
Proceedings of Performance '87, Brussels, Belgium, December 1987.

[43] Man-Tung Tony Hsiao.
Optimal Decentralized Flow Control in Computer Communication Networks.
PhD thesis, Center for Telecommunications Research, Columbia University, New York, NY. 86.

[44] Chi-Yin Huang Hsu and Jane W.-S Liu.
Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems.
Proceedings of 6th IEEE Distributed Computer Systems Conference, 1986.

[45] P.H. Hughes and G. Moe.
A Structural Approach to Computer Performance.
Analysis Proceedings AFIPS 1973 Spring Joint Computer Conference, AFIPS Press, Arlington, VA, 1973.

[46] K. Hwang et al.
Engineering Computer Network (ECN): A Hard Wired Network of Unix Computer Systems.
Proceedings of the National Computer Conference, May 1981.

[47] K. Hwang et al.
A Unix Based Local Computer Network with Load Balancing.
IEEE Computer, April 1982.

[48] Jeffrey Jaffe.
Bottleneck Flow Control.
IEEE Transactions on Communication, COM-29(7), July 1981.

[49] Jeffrey Jaffe.
Flow Control Power is Nondecentralizable.
IEEE Transactions on Communication, COM-29(9), September. 1981.

[50] R. E. Kahn.
The Organization of Computer Resources into a Packet Radio Network.
IEEE Transactions on Communications, COM-25(1), January 1977.

[51] Leonard Kleinrock.
Queueing Systems: Volume I: Theory.
Wiley Interscience, New York, 1975.

[52] Leonard Kleinrock.
Queueing Systems: Volume II: Computer Applications.
Wiley Interscience, New York, 1975.

[53] Phillip Krueger and Raphael A. Finkel.
An Adaptive Load Balancing Algorithm for a Multicomputer.
University of Wisconsin, Madison 539, April 1984.

[54] Phillip Krueger and Miron Livny.
When it the Best Load Sharing Algorithm a Load Balancing Algorithm?.
University of Wisconsin, Madison, April 1987.

[55] James F. Kurose.
Time-Constrained Communication in Multiple Access Networks.
PhD thesis, Columbia University, 1984.

[56] James F. Kurose, Mischa Schwartz, and Yechiam Yemini.
Distributed Multiple Access Protocols and Real-Time Communication.
Computing Surveys, 6(1), March 1984.

[57] James F. Kurose, Mischa Schwartz, and Yechiam Yemini.
A Microeconomic Approach to Optimization of Channel Access Policies in Multiaccess Networks.
Proceedings of the 5th IEEE Distributed Computer Systems Conference, pages 70-80, May 1985.

[58] James F. Kurose and Rahul Simha.
A Microeconomic Approach to Optimal File Allocation.
Proceedings of the 6th IEEE Distributed Computer Systems Conference, 1986.

[59] James F. Kurose and Rahul Simha.
Second Derivative Algorithms for Optimal Resource Allocation in Distributed Computer Systems.
Proceedings of the 7th International Conf. on Distributed Computing Systems, 1987.

[60] James F. Kurose and Rahul Simha.
A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems.
Univ. of Mass. at Amherst, 1987.
Submitted to IEEE Transactions on Computers.

[61] L. Lamport, R. Shostak, and M. Pease.
The Byzantine Generals Problem.
ACM Transactions on Programming Languages and Systems, 4(3):382-401, 1982.

[62] K.D. Levin and H.L. Morgan.
Optimizing Distributed Databases - A Framework for Research.
Proceedings AFIPS 1975 National Computer Conference, 44, 1975.

[63] S. Li and T. Basar.
Distributed Algorithms for the Computation of Noncooperative Criteria.
Automatica, 23(4), July 1987.

[64] Miron Livny.
The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems.
PhD thesis, Weizmann Institue of Science, Rehovot, Israel. 1984.

[65] Miron Livny and Myron Melman.
Load Balancing in Homogeneous Distributed Systems.
Proceedings of the ACM Computer Networking Performance Symposium, pages 47-55, April 1982.

[66] Thomas W. Malone, Richard E. Fikes, Kenneth R. Grant, and Michael T. Howard.
Market Like Load Sharing in Distributed Computing Environments.
MIT, 1987.