LOAD BALANCING IN DISTRIBUTED SYSTEMS:
A GAME THEORETIC APPROACH

APPROVED BY SUPERVISING COMMITTEE:

Dr. Anthony T. Chronopoulos, Supervising Professor

Dr. Ming-Ying Leung, Co-Supervising Professor

Dr. Rajendra Boppana

Dr. Turgay Korkmaz

Dr. Chia-Tien Dan Lo

Accepted:

Dean of Graduate Studies

Dedication

This dissertation is dedicated to my wife, Sanda Grosu and to my daughter, Ioana Grosu.

LOAD BALANCING IN DISTRIBUTED SYSTEMS:
A GAME THEORETIC APPROACH

by

DANIEL GROSU, M.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2003

Acknowledgements

I would like to express sincere gratitude for the time, encouragement, and guidance provided by my advisors, Dr. Anthony Chronopoulos and Dr. Ming-Ying Leung, whose help cannot be truly expressed in words. I also would like to thank Dr. Rajendra Boppana, Dr. Turgay Korkmaz and Dr. Chia-Tien Dan Lo for serving on my dissertation committee and for their support and suggestions. I am grateful to Dr. John Nash, for his influence and enlightening discussions on game theory.

I am grateful to my parents Constantin and Gherghina, my wife Sanda and my daughter Ioana for their absolute love, devotion and support in everything.

I would also like to thank all of my friends I made during the past four years on my path toward earning my Ph.D. Thanks for all the great times and for all the encouragement and suggestions. It made this journey much more pleasant and meaningful. Thank you all.

This research was supported in part by the NASA grant NAG 2-1383 (1999-2001), Texas Advanced Research/Advanced Technology Program grant ATP 003658-0442-1999, a grant from UTSA Center for Information Assurance and Security and the University of Texas at San Antonio Department of Computer Science. The research for this dissertation was conducted at the University of Texas at San Antonio within the laboratories of the Department of Computer Science.

May 2003

LOAD BALANCING IN DISTRIBUTED SYSTEMS: A GAME THEORETIC APPROACH

Daniel Grosu, Ph.D.
The University of Texas at San Antonio, 2003

Supervising Professor: Dr. Anthony T. Chronopoulos

In this dissertation we introduce and investigate a new generation of load balancing schemes based on game theoretic models. First, we design a load balancing scheme based on a cooperative game among computers. The solution of this game is the Nash Bargaining Solution (NBS) which provides a Pareto optimal and fair allocation of jobs to computers. The main advantages of this scheme are the simplicity of the underlying algorithm and the fair treatment of all jobs independent of the allocated computers. Then we design a load balancing scheme based on a noncooperative game among users. The Nash equilibrium provides a user-optimal operation point for the distributed system and represents the solution of the proposed noncooperative load balancing game. We present a characterization of the Nash equilibrium and a distributed algorithm for computing it. The main advantages of our noncoopeartive scheme are its distributed structure and user-optimality. We compare the performance of the proposed load balancing schemes with that of other existing schemes and show their main advantages.

This dissertation is also concerned with the design of load balancing schemes for distributed systems in which the computational resources are owned and operated by different self interested agents. In such systems there is no a-priori motivation for cooperation and the agents may manipulate the resource allocation algorithm in their own interest leading to severe performance degradation and poor efficiency. Using concepts from mechanism design theory (a sub-field of game theory) we design two load balancing protocols that force the participating agents to report their true parameters and follow the rules. We prove that our load balancing protocols are truthful and satisfy the voluntary participation condition. Finally we investigate the effectiveness of our protocols by simulation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Load balancing in distributed systems

Distributed systems offer the potential for sharing and aggregation of different resources such as computers, storage systems and other specialized devices. These resources are distributed and possibly owned by different agents or organization. The users of a distributed system have different goals, objectives and strategies and their behavior is difficult to characterize. In such systems the management of resources and applications is a very complex task.

A distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases the load balancing problem becomes important. The purpose of load balancing is to improve the performance of a distributed system through an appropriate distribution of the application load. A general formulation of this problem is as follows: given a large number of jobs, find the allocation of jobs to computers optimizing a given objective function (e.g. total execution time).

One way to deal with the management of distributed systems is to have an unique decision maker that will lead the system to its optimum. This is the approach used in most of the existing load balancing solutions [67]. Another way is to let the users to cooperate in making the decisions such that each user will operate at its optimum. The users have complete freedom of preplay communication to make joint agreements about their operating points. Cooperative game theory provides a suitable modeling framework for investigating such cooperative settings [50].

Another approach is letting the users to compete for resources and allow them to reach an equilibrium where each of them acquires her optimum. In other words the equilibrium is the setting where no user can increase her payoff by unilaterally deviating from the equilibrium. This kind of equilibrium is called the Nash equilibrium. Game theory offers a viable modeling paradigm for studying this problem [50]. Using this approach we can obtain solutions in which the decision process is distributed and scalable. Moreover, system's utilization is improved and the fairness of allocation is guaranteed.

There is a large body of literature on load balancing and all these studies can be broadly characterized as static and dynamic. Static load balancing uses a priori knowledge of the applications and statistical information about the system. Dynamic load balancing base their decision making process on the current state of the system. A good load balancing scheme needs to be general, stable, scalable, and to add a small overhead to the system. These requirements are interdependent, for example a general load balancing scheme may add a large overhead to the system, while an application specific load balancing scheme may have a very small overhead.

Recently, with the emergence of the Internet as a global platform for computation and communication, the need for new load balancing schemes capable to deal with self interested participants has increased. This leads to the need of mechanism design based schemes (to be explained in later chapters) which motivate the participants to report their true parameters and follow the load allocation algorithm. In addition these new schemes must be scalable and have a very little communication overhead.

1.2 Contributions of this dissertation

In this dissertation we introduce and investigate a new generation of load balancing schemes based on game theoretic models. The motivation for a game theoretic approach to load balancing is twofold. First, the computational resources are distributed and controlled by many users having different requirements. Second, the users are likely to behave in a selfish manner and their behavior cannot be characterized using conventional techniques and models. Game theory provides a suitable framework for characterizing such settings.

Our first goal is to find a formal framework for characterization of fair allocation schemes that are optimal for each job. We formulate the load balancing problem for single class job distributed systems as a cooperative game among computers and we provide a method for computing the solution. We prove that the allocation obtained using our scheme is fair and optimal for each job. We compare our cooperative scheme with other existing schemes (which are not based on game theoretic models). The main advantages of our cooperative scheme are the simplicity of the underlying algorithm and the fair treatment of all jobs independent of the allocated processors.

Another approach is to use a noncooperative game as a modeling framework for load balancing. Using this framework we formulate the load balancing problem in distributed systems as a noncooperative game among users. The Nash equilibrium provides a user-optimal operation point for the distributed system. We give a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compare the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system and the decision process is distributed and thus scalable.

The problem of scheduling and load balancing in distributed systems has been extensively investigated under the assumption that the participants (e.g. computers, users) are truthful and follow a given algorithm. In current distributed systems the computational resources are owned and operated by different agents or organizations. In such systems there is no a-priori motivation for cooperation and the agents may manipulate the resource allocation algorithm in their own interest leading to severe performance degradation and poor efficiency. Solving such problems involving selfish agents is the object of *mechanism design theory*. This theory helps design protocols in which the agents are always forced to tell the truth and follow the rules. Such mechanisms are called *truthful* or *strategy-proof*.

We design a truthful mechanism for solving the static load balancing problem in heterogeneous distributed systems. We prove that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. Based on the proposed mechanism we derive a load balancing protocol and we study its effectiveness by simulations.

We also study the problem of designing load balancing mechanisms with verification. An agent may report a value (bid) different from its true value. Here the true value characterizes the actual processing capacity of each computer. In addition, an agent may choose to execute the jobs allocated to it with a different processing rate given by its execution value. Thus, an agent may execute the assigned jobs at a slower rate than its true processing rate. The goal of a truthful mechanism with verification is to give incentives to agents such that it is beneficial for them to report their true values and execute the assigned jobs using their full processing capacity. This assumes that the mechanism knows the execution values after the jobs were completed. We design a truthful mechanism with verification that solves the load balancing problem and satisfies the voluntary participation condition. We study the effectiveness of our load balancing mechanism by simulation.

1.3 Organization of the dissertation

This rest of the dissertation is organized as follows. In Chapter 2 we present a brief review of some game theory concepts and a survey of existing load balancing schemes. Chapter 3 describes our load balancing cooperative game. Chapter 4 presents our noncooperative load balancing game for multi-class job distributed systems. Chapter 5 presents our work on algorithmic mechanism design for load balancing. Chapter 6 presents a load balancing mechanism with verification. Chapter 7 summarizes the dissertation and presents possible directions for future work.

Chapter 2

Background

In this chapter we review the basics of game theory and we present an extensive survey of existing load balancing schemes in distributed systems.

2.1 Game Theory Concepts

A finite game can be characterized by three elements: the set of players $i \in \mathcal{I}, \mathcal{I} = \{1, 2, \ldots, I\}$; the pure strategy space S_i for each player i; and the objective functions $p_i : S_1 \times \ldots \times S_I \rightarrow \mathbf{R}$ for each player $i \in \mathcal{I}$. Let $s = (s_1, s_2, \ldots, s_I)$ be a *strategy profile* $s \in S$, where $S = S_1 \times S_2 \times \ldots \times S_I$. Each player objective is to minimize her own objective function. In economics the objective functions are usually firm's profit and each user wants to maximize them, while in our context the objective functions usually represent job execution times.

There are two main classes of games: *cooperative games* in which the players have complete freedom of preplay communications to make joint agreements and *noncooperative games* in which no preplay communication is permitted between the players [93].

If the interaction between users occurs only once, the game is called *static* and if the interaction occurs several times the game is called *dynamic*. A static game played many times is called a *repeated game*. A game in which one user (the leader) imposes its strategy on the other self optimizing users (followers) is called *Stackelberg game* [50].

One of the cornerstones of noncooperative game theory is the notion of *Nash equilibrium* [102, 103].

In essence a Nash equilibrium is a choice of strategies by the players where each player's strategy is a best response to the other player's strategies.

If we consider that each player's goal is to minimizes her objective function then the Nash equilibrium can be defined as follows. A strategy profile s^* is a Nash equilibrium if for all players $i \in \mathcal{I}$:

$$p_i(s_1^*, s_2^*, \ldots, s_i^*, \ldots, s_I^*) \leq p_i(s_1^*, s_2^*, \ldots, s_i, \ldots, s_I^*) \quad \text{for all} \quad s_i \in S_i \tag{2.1}$$

In other words, no player can decrease the value of its objective function by unilaterally deviating from the equilibrium.

If we consider that each player's goal is to maximize her objective function, then the above equation changes to:

$$p_i(s_1^*, s_2^*, \ldots, s_i^*, \ldots, s_I^*) \geq p_i(s_1^*, s_2^*, \ldots, s_i, \ldots, s_I^*) \quad \text{for all} \quad s_i \in S_i \tag{2.2}$$

Note that in all the examples in this chapter players maximize their objective functions whereas in our models players minimize their objective functions.

One difficulty is the lack of uniqueness of Nash equilibria for some games. Rosen [115] showed that if the game is concave the equilibrium point is unique. Nash equilibria are consistent predictions of how the game will be played in the sense that if all players predict that a particular Nash equilibrium will occur then no player has an incentive to play differently [50].

When some players do not know the objective functions of the others the game is said to have *incomplete information* and the corresponding Nash equilibria are called *Bayesian equilibria* [57,58]. This type of game is also called *Bayesian game*. If the objective functions are common knowledge the game is said to have *complete information*.

For more details on game theory, refer to [11, 50]. In the following we present three classical games to illustrate the concepts presented above.

Example 2.1

The most well-known game is the *Prisoners' Dilemma* [50]. The following is one of variant of the story behind the game. A crime has been committed and two people are arrested. The police put the suspects in different cells to prevent communication between them. The police questions the prisoners. If both prisoners confess they will be punished but not severely and the associated payoff of this outcome is 1. If only one prisoner confesses, he will go to prison (associated payoff -1) while the other will be freed (associated payoff 2). If both players defect, both will receive a less severe punishment than a sole confessor receives (associated payoff 0). Each player's objective is to maximize his own payoff.

This game is a static two player game and can be described by the payoff matrix shown on Figure 2.1. The strategies of player 1 are represented as row labels and those of player 2 as column labels. The payoffs are placed on each matrix entry as a pair where the first number is the payoff of player 1 and the second number is the payoff of player 2.

		player 2	
		C	D
player 1	C	1, 1	−1, 2
	D	2, −1	0, 0

Figure 2.1. The prisoners' dilemma game.

This game is called Prisoners' Dilemma because the rational outcome is (D, D) that gives suboptimal payoffs of (0, 0) instead of optimal ones (1, 1). Let's analyze the rational outcome of the game and show that a rational player will play D. If player 1 plays C, then player 2 will obtain a better payoff ($2 > 1$) by playing D than playing C. If player 1 plays D then player 2 will obtain a better payoff ($0 > -1$) playing D than playing C. Thus, independent of the strategy of player 1, player 2 will be better off by playing D. Symmetrically, a rational player 1 will be better playing D independent of the strategy of player 2. The outcome of the game played by rational players will be (D, D), yielding the payoffs (0, 0).

Using the definition of a finite game presented at the beginning of this section we can characterize the Prisoners' Dilemma game as follows:

- The set of players $\mathcal{I} = \{1, 2\}$.

- The strategy spaces for both players are: $S_1 = S_2 = \{\mathrm{C}, \mathrm{D}\}$
- The objective functions (p_1, p_2) are:

$$\begin{array}{ll} p_1(\mathrm{C},\mathrm{C}) = 1 & p_2(\mathrm{C},\mathrm{C}) = 1 \\ p_1(\mathrm{C},\mathrm{D}) = -1 & p_2(\mathrm{C},\mathrm{D}) = 2 \\ p_1(\mathrm{D},\mathrm{C}) = 2 & p_2(\mathrm{D},\mathrm{C}) = -1 \\ p_1(\mathrm{D},\mathrm{D}) = 0 & p_2(\mathrm{D},\mathrm{D}) = 0 \end{array}$$

In this game (D, D) is a Nash equilibrium because: if player 1 plays D, the best response of player 2 is to play D; and if player 2 plays D the best response of player 1 is to play D. □

Example 2.2

We present another classical game for which there exists two Nash equilibria. This game is called *The Battle of the Sexes* [50]. The story behind this game is that a man and a woman wish to go to an event together but disagree about whether to go to a football game (strategy F) or to a ballet (strategy B). Both prefer to be together rather than go alone. The payoff matrix is presented in Figure 2.2.

		player 2	
		B	F
player 1	B	2, 1	0, 0
	F	0, 0	1, 2

Figure 2.2. The battle of the sexes game.

This game has two Nash equilibria: (B, B) and (F, F). The reasoning is the following. If the woman plays B, then the best response of the man is B, and if the man plays B the best response of the woman is B. Similarly if the woman plays F the best response of the man is F, and if the man plays F the best response of the woman is F. □

Example 2.3

The *Envelope game* [54] is a classical example of Bayesian game in which the payoffs depend on the probable state of the ”outside world”. The story is that a father offers each of his two sons an envelope with either \$ 10^m or \$ 10^n. m and n satisfy the following conditions: $0 \leq m, n \leq 6$ and $|m - n| = 1$. Each player(brother) has two strategies: he can accept the envelope (No Bet) or engage in a bet (Bet) in which he pays the father \$1 for the right to swap the envelopes with his brother. The envelopes will be swapped only if both brothers will chose to bet. If only one brother chooses to bet the envelopes will not be swapped and he will lose \$1. If we denote the state of the outside world with (m, n) and assume that m (n) is the exponent of the payoff for the first (second) player the payoff matrix can be represented as in Figure 2.3.

		player 2	
		B	NB
player 1	B	$10^n-1, 10^m-1$	$10^m-1, 10^n$
	NB	$10^m, 10^n-1$	$10^m, 10^n$

Figure 2.3. The envelope game.

Here the Bayesian equilibrium will be (NB, NB). For example if both players are certain that the state of the world is (2, 3) they will play as follows. Player 2 best strategy is NB because he got the envelope containing the greatest amount of money. Player 1 best strategy is also NB because player 2 will not bet. Otherwise he will lose \$1 given that player 2 plays NB. □

2.2 Load Balancing

In a distributed system it is possible for some computers to be heavily loaded while others are lightly loaded. This situation can lead to poor system performance. The goal of load balancing is improving the performance by balancing the loads among computers.

There are two main categories of load balancing policies: *static policies* and *dynamic policies* [21, 114]. Static policies base their decision on statistical information about the system. They do not take into consideration the current state of the system. Dynamic policies base their decision on the current state of the system. They are more complex than static policies.

2.2.1 Static Load Balancing

Among static policies we can distinguish between *distributed policies* and *centralized policies*. In a distributed policy the work involved in making decisions is distributed among many decision makers. In a centralized policy we may have only one decision maker or the common decision of many cooperating decision makers is made in a centralized way.

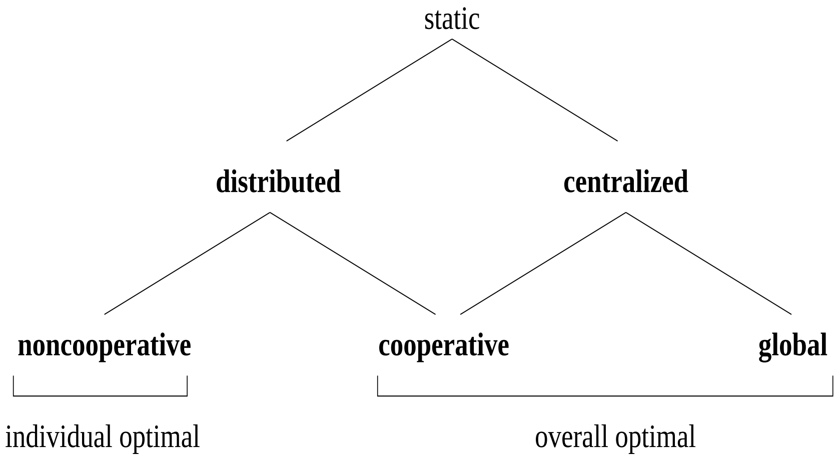

Figure 2.4. A taxonomy for static load balancing policies.

Within the realm of distributed policies we can distinguish between *cooperative policies* which involve cooperation between decision makers (e.g. users, processors) and *noncooperative policies* where decision makers make their decisions noncooperatively.

In the cooperative case all decision makers are working toward a common system wide goal. The system wide goal is to optimize some criterion function. System wide goal examples are: minimizing total process execution time, maximize utilization of system resources or maximize system throughput. In this case we can have *optimal policies* in the situation in which we can feasible obtain an optimum for the criterion function and *suboptimal policies* when obtaining the optimum is computationally infeasible. Cooperative game theory offers a viable framework for studying the cooperative policies.

In the noncooperative case all decision makers compete with one another and they eventually settle to an equilibrium state where each of them reaches its optimum (individual optimum). Each decision maker may have different objective, some of them contradictory. Game theory offers a viable modeling paradigm for investigating this problem. The important aspect of noncooperative policies is that we can obtain solutions in which the decision process is distributed and thus scalable.

We propose an extension of Casavant's taxonomy for static load balancing [21], by adding two new hierarchical levels: distributed-centralized and cooperative-noncooperative-global (Figure 2.4).

Often, jobs in a distributed system can be divided into different classes based on their resource usage characteristics and ownership. For example the jobs that belong to a single user can form a class. Also, we can distinguish different classes of jobs by their execution times. Depending on how many job classes are considered we can have single class or multi-class job distributed systems.

In the following we present different approaches to static load balancing problem for both single class and multi-class job systems.

I. Static Load Balancing for Single Class Job Systems: Models and Problem Formulation

There are two typical optima, depending on the degree of cooperation among decision makers (e.g. users, jobs) [66, 68]:

I. A) *Overall optimum (social optimum) schemes:* We can obtain the overall optimum using a global or a cooperative policy. In the case of global policies there is only one decision maker that optimizes the expected response time of the entire system over all jobs. This is the classical approach (i.e. is not based on game theoretic models) and has been studied extensively under various assumptions [15, 84, 104, 116]. There are no known studies of cooperative policies for static load balancing. In Chapter 3 we propose and investigate a cooperative scheme for load balancing in single class job systems.

I. B) *Wardrop equilibrium (individual optimum) schemes:* This equilibrium can be realized by a distributed noncooperative policy. Each of infinitely many users optimizes its own expected response time independently of others. This can be viewed as a noncooperative game among jobs. At the equilibrium point a job cannot receive any further benefit by changing its own decision. It is assumed that the decision of a job has a negligible impact on the performance of other individuals.

I. A) Overall optimum schemes

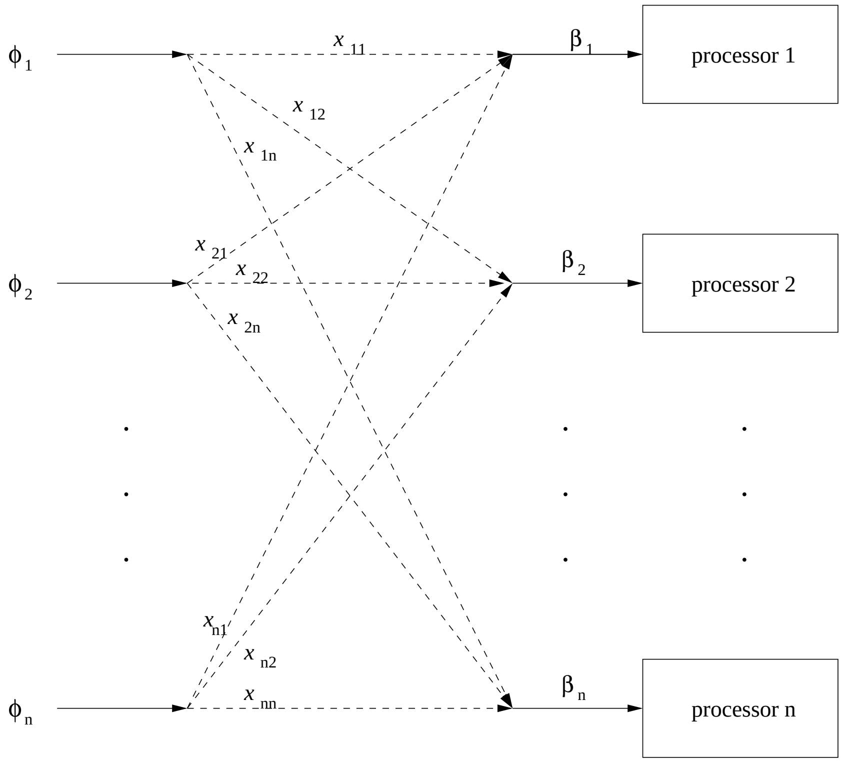

Figure 2.5. Single class job model.

All existing overall optimum schemes are not based on game theoretic models and can be classified as global schemes. Tantawi and Towsley [128] studied load balancing of single class jobs in a distributed heterogeneous system connected by single channel networks. The computers are modeled as a central server and the communication network is modeled as M/G/∞ and M/M/1 queueing systems [73]. Jobs arrive at computer i, $i = 1 \ldots, n$ according to a time invariant Poisson process. Their model is presented in Figure 2.5.

They used the following notations:

β_i - job processing rate at computer i, to be determined by the load balancing scheme;

$\beta = [\beta_1, \beta_2, \ldots, \beta_n]$;

ϕ_i - external job arrival rate at computer i;

Φ - total external arrival rate ($\Phi = \sum_{i=1}^{n} \phi_i$);

λ - network traffic;

x_{ij} - job flow rate from computer i to computer j;

$F_i(\beta_i)$ - expected execution delay of job processed at computer i (increasing positive function);

$G(\lambda)$ - source-destination independent expected communication delay (nondecreasing function);

$D(\beta)$ - expected response time of a job;

Remark: The parameters Φ and ϕ_i are fixed.

The problem is to minimize:

$$D(\beta) = \frac{1}{\Phi}[\sum_{i=1}^{n} \beta_i F_i(\beta_i) + \lambda G(\lambda)] \tag{2.3}$$

subject to the following constraints:

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{2.4}$$

$$\beta_i \geq 0, \qquad i = 1, 2, \ldots, n \tag{2.5}$$

where the network traffic λ may be expressed in terms of β_i:

$$\lambda = \frac{1}{2} \sum_{i=1}^{n} |\phi_i - \beta_i| \tag{2.6}$$

Based on this nonlinear optimization problem, Tantawi and Towsley [128] proposed two algorithms for computing the optimal load of each computer. They also studied the properties of the optimal solution.

Kim and Kameda [72] derived a more efficient algorithm to compute the loads of each computer. Li and Kameda proposed a similar algorithm for load balancing in star and tree network configurations [88, 89].

They also proposed an algorithm for general networks in [90]. Tang and Chanson [127] proposed and studied several optimal schemes that take into account the job dispatching strategy.

I. B) Wardrop equilibrium schemes

Kameda *et al.* [67] studied the Wardrop equilibrium. They used an equivalent problem for expressing the individual optimization problem. This equivalence was studied in transportation science [95].

Let $F_i^*(\beta)$ and $G^*(\lambda)$ be the expected processing and communication delays for the equivalent problem.

$$F_i^*(\beta_i) = \frac{1}{\beta_i}\int_0^{\beta_i} F_i(\beta_i^{'})d\beta_i^{'}, \qquad F_i^*(0) = F_i(0) \tag{2.7}$$

$$G^*(\lambda) = \frac{1}{\lambda}\int_0^{\lambda} G(\lambda^{'})d\lambda^{'}, \qquad G^*(0) = G(0) \tag{2.8}$$

The equivalent optimization problem can be formulated as follows:

minimize:

$$D^*(\beta) = \frac{1}{\Phi}[\sum_{i=1}^{n} \beta_i F_i^*(\beta_i) + \lambda G^*(\lambda)] \tag{2.9}$$

subject to the following constraints:

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{2.10}$$

$$\beta_i \geq 0, \qquad i = 1, 2, \ldots, n \tag{2.11}$$

where the network traffic may be expressed in terms of β_i:

$$\lambda = \frac{1}{2}\sum_{i=1}^{n} |\phi_i - \beta_i| \tag{2.12}$$

This problem is similar with the previous one except for the differences between $F_i^*(\beta_i)$, $G^*(\lambda)$ and $F_i(\beta_i)$, $G(\lambda)$. The authors used a variant of the algorithm for problem I. A) to find the loads of each computer such that the Wardrop equilibrium is reached [67].

II. Static Load Balancing for Multi-Class Job Systems: Models and Problem Formulation

As in the case of single class job problem we have two typical optima depending on the degree of cooperation among decision makers [66, 68]:

II. A) *Overall optimum (social optimum) schemes:* We can obtain the overall optimum using a global or a cooperative policy. In the case of global policies there is only one decision maker that optimizes the expected response time of the entire system over all jobs. This is the classical approach (i.e. is not based on game theoretic models) and has been studied extensively [34, 104]. There are no known studies of cooperative policies for static load balancing. All the studies involved the global approach.

II. B) *Nash equilibrium (class optimum) schemes:* This equilibrium can be obtained by a distributed noncooperative policy. Infinitely many jobs are grouped into a finite number (K) of classes ($K > 1$). Each class has its own decision maker and the goal is to minimize the expected response time of all the jobs in the class. This can be viewed as a noncooperative game among classes. At the equilibrium, each class cannot receive any further benefit by changing its decision. The equilibrium is called *class optimum* or *Nash equilibrium*.

Remark: When the number of classes becomes one the Nash equilibrium reduces to the overall optimum. When the number of classes becomes infinitely many the Nash equilibrium reduces to the Wardrop equilibrium.

II. A) Overall optimum schemes

All existing overall optimum schemes are not based on game theoretic models. In the following we present the model proposed by Kim and Kameda [71]. Their model is presented in Figure 2.6. They have considered the problem of static load balancing of multi-class jobs in a distributed system. The distributed system consists of a set of heterogeneous computers connected by a single channel communication network. A key assumption here is that the network delay does not depend on the source-destination pair (e.g. Ethernet).

They used the following notations:

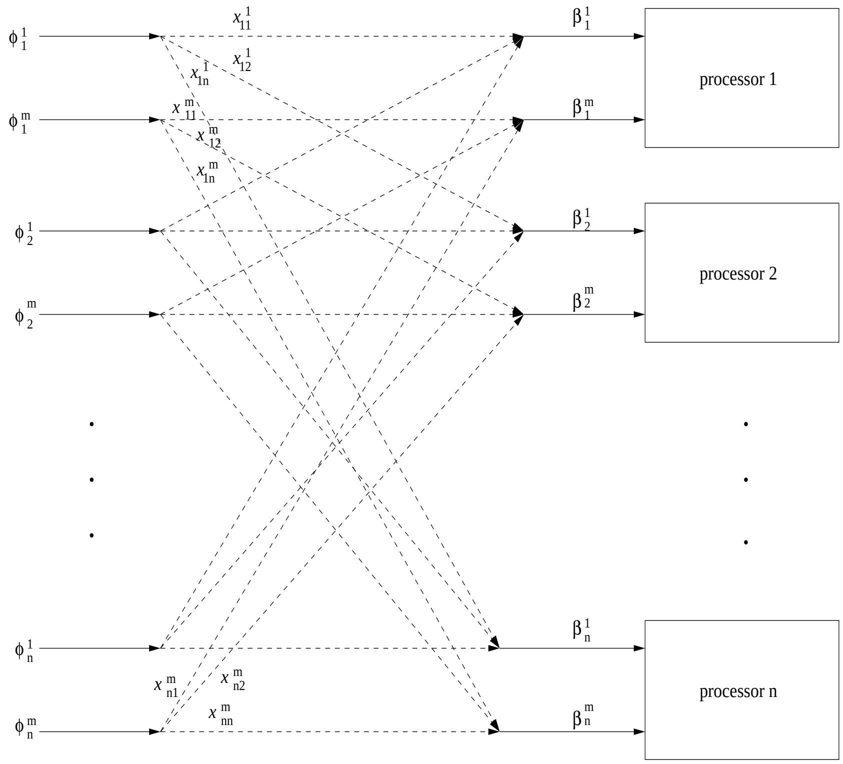

Figure 2.6. Multi-class job model.

n - number of computers;

m - number of job classes;

ϕ_i^k - class k job arrival rate to computer i;

x_{ij}^k - class k job flow rate from computer i to computer j;

β_i^k - class k job processing rate at computer i, to be determined by the load balancing scheme. ($\beta_i^k = \sum_{l=1}^n x_{li}^k$);

λ^k - class k job traffic through network;

ϕ^k - Total class k job arrival rate. $\phi^k = \sum_{i=1}^n \phi_i^k$;

Φ - Total external job arrival rate. $\Phi = \sum_{k=1}^m \phi_i^k$;

$\beta_i = [\beta_i^1, \beta_i^2, \ldots, \beta_i^m]$;

$\beta = [\beta_1, \beta_2, \ldots, \beta_n]$;

$\lambda = [\lambda^1, \lambda^2, \ldots, \lambda^m]$;

$F_i^k(\beta_i)$ - expected processing delay of class k job processed at computer i (differentiable and convex function);

$G^k(\lambda)$ - expected communication delay of class k job (differentiable, nondecreasing and convex function);

$D(\beta)$ - system-wide expected response time;

Remark: Φ, ϕ^k and ϕ_i^k are fixed parameters.

Minimizing the system-wide expected response time corresponds to the following optimization problem: minimize:

$$D(\beta) = \frac{1}{\Phi}\sum_{k=1}^{m}[\sum_{i=1}^{n}\beta_i^k F_i^k(\beta_i) + \lambda^k G^k(\lambda)] \tag{2.13}$$

subject to:

$$\sum_{i=1}^{n}\beta_i^k = \phi^k, \qquad k = 1, 2, \ldots, m \tag{2.14}$$

$$\beta_i^k \geq 0, \qquad i = 1, 2, \ldots, n,\ k = 1, 2, \ldots, m \tag{2.15}$$

where the class k network traffic may be expressed in terms of β_i^k:

$$\lambda^k = \frac{1}{2}\sum_{i=1}^{n}|\phi_i^k - \beta_i^k| \tag{2.16}$$

An algorithm for computing the loads (β_i^k) was proposed in [71].

II. B) Nash equilibrium schemes

There are no known Nash equilibrium schemes for load balancing in multi-class job distributed systems. Kameda *et al.* [67] proposed a load balancing scheme based on a noncooperative policy but it does not obtain the Nash equilibrium. In Chapter 4 we derive a load balancing scheme that obtains the Nash equilibrium.

2.2.2 Dynamic Load Balancing

Dynamic policies base their decision on the current state of the system. Despite the higher runtime complexity dynamic policies can lead to better performance than static policies. There are two main classes of dynamic load balancing policies: *centralized* and *distributed* [21, 114, 117].

Centralized policies:

In this type of policies a dedicated computer is responsible for maintaining a global state of the system [14, 16, 63]. Based on the collected information the central computer makes allocation decisions. For a large number of computers the overhead of such schemes become prohibitive and the central decision computer becomes the bottleneck.

We classify the studies of centralized schemes based on the underlying model used for analysis. There are two categories of models: queueing models and non-queueing models.

- **Queueing models:**

  Chow and Kohler [24] proposed and studied three load balancing centralized schemes. They introduced an approximate numerical method for analyzing two-processor heterogeneous systems.

  Banawan and Zahorjan [10] use semi-Markov decision processes to show that an optimal centralized policy that uses the instantaneous queue length independent of the system utilization does not exist.

  Bonomi and Kumar [14, 16] showed that minimizing the expected job response time is equivalent to balancing the processor idle times in a weighted least squares sense. They used this result to develop a dynamic centralized policy that utilizes processor idle time measurements which are sent periodically to the central job scheduler.

  Bonomi [13] proposed an approach to design centralized load balancing policies and showed that it produces a solution performing better than the join-the-shortest-queue policy.

  Shenker and Weinrib [122] considered a finite number of fast computers and an infinite number of slow computers and showed via simulation that for heavy load it is better never to queue a job at a fast computer, but use a slow computer.

Avritzer *et al.* [9] concluded that the load balancing is effective even in the presence of high loads if the parameters of the algorithms are carefully tuned. Based on their findings they derived a self tuning centralized load balancing algorithm.

Lin and Raghavendra [91] proposed a dynamic load balancing policy with a centralized job scheduler that uses global state information in making decisions. They studied the performance of this policy for systems with non-negligible job transfer delays.

Goswami *et al.* [53] proposed a predictive strategy that uses a statistical pattern-recognition method to predict the resource requirements of a process before its execution.

Cai *et al.* [19] studied by simulation the influence of parallel and sequential workloads on centralized load balancing decisions.

Mitzenmacher [100] studied several dynamic randomized load balancing schemes. It is shown that allowing tasks two choices instead of one leads to exponential improvement in the expected time a task spends in the system.

- **Non-queueing models:**

  Ahmad and Ghafoor [2] proposed a hierarchical approach for load balancing in which the system is partitioned into several regions. Each region has a central scheduler that optimally schedules tasks and maintains state information.

  Kulkarni and Sengupta [80] proposed a centralized load balancing scheme based on differential load measurement. The selection of node pairs for load transfer is based on differential load information.

  Han *et al.* [56] considered the load balancing problem in distributed systems equipped with circuit or cut-through switching capability. They provide several algorithms to solve the load balancing problem on these kinds of systems.

  Gil [52] presented a $O(\log \log n)$ time load balancing algorithm which achieves an almost flat distribution of tasks for each processor in a PRAM model.

  Hui and Chanson [63] proposed two centralized strategies: one that takes network delay into account to avoid errors in scheduling jobs; the other delays job execution when the system is fully used.

Distributed policies:

In this type of policies each computer constructs its own view of the global state [20, 27, 78, 132]. Most of the distributed schemes are suboptimal and heuristic due to unavailability of accurate and timely global information. A distributed dynamic scheme has three components: 1) a *location policy* that determines which computers are suitable to participate in load exchanges; 2) an *information policy* used to disseminate load information among computers; and 3) a *transfer policy* that determines whether task transfer activities are needed by a computer. Distributed load balancing schemes can be classified as: *sender-initiated*, *receiver-initiated* and *symmetrically-initiated*.

a) **Sender-initiated schemes:**

In these schemes the highly loaded processors dispatch their jobs to lightly loaded processors. A processor is identified as a sender if a new originating task at the processor makes the queue length exceed a threshold [123]. Eager *et al.* [38] studied three simple, fully distributed sender-initiated schemes. These three schemes differ only in their location policies referred to as *Random*, *Threshold* and *Shortest*. With the *Random* policy a destination processor is selected at random and the task is transferred to that processor. Under the *Threshold* policy the sender repeatedly selects and probes at random (up to *probe limit*) another processor and if its load is found under the threshold a task is transferred to that processor. Under the *Shortest* policy a number of processors (probe limit) are selected at random and probed to determine their queue length. The processor with the shortest queue is selected as the destination for task transfer. The performance of *Shortest* is not significantly better than that of *Threshold* indicating that using more detailed state information does not necessarily improve performance significantly. All the three policies lead to system instability at high loads. Sender-initiated policies perform better than receiver-initiated policies at low to moderate loads.

Dandamudi [33] studied the sensitivity of both sender- and receiver-initiated policies to variances in inter-arrival and service times along with the processor scheduling policy.

b) **Receiver-initiated schemes:**

In these schemes the lightly loaded processors request jobs from the busy processors. The load balancing action is initiated when a task departs and if the queue length falls below a threshold (the processor is identified as a receiver) [123]. Eager *et al.* [37] studied a receiver-initiated scheme. The receiver repeatedly selects and probes at random (up to *probe limit*) another processor and if its load is found above the threshold and the processor is not already in the process of task transferring, a task is transferred from that processor. It was found that receiver-initiated schemes are preferable at high system loads.

Kumar *et al.* [81] studied the scalability of receiver- and sender-initiated schemes for different architectures such as hypercube, mesh and network of workstations. They derived lower bounds on the scalability for each architecture.

Schaar *et al.* [120] proposed an analytical and simulation model that includes the effect of communication delays on load balancing schemes. Based on this model they derived one receiver- and one sender-initiated schemes that take into account the communication delay in making load balancing decisions.

Liu and Silvester [92] derived an approximate performance model for analyzing load dependent queueing systems and applied it to receiver-initiated schemes.

Willebeek-Le Mair and Reeves [133] proposed and studied a receiver- and a sender-initiated diffusion algorithm. For each algorithm they studied the tradeoff between the accuracy of load balancing decision and the overhead incurred by the balancing process.

c) **Symmetrically-initiated schemes:**

These schemes are a combination of previous two schemes where both senders and receivers initiate the load balancing process. Based on the current load level of a processor a symmetrically-initiated scheme switches automatically between a sender-initiated policy when the load goes above the threshold and a receiver-initiated when the load level drops below the threshold. These schemes have the advantages of both sender- and receiver-initiated schemes. At low loads the sender-initiated part is more effective in finding underloaded processors, whereas at high loads the receiver initiated part is more effective in finding overloaded processors [123].

Kruger and Shivaratri [79] proposed an adaptive symmetrically-initiated scheme which is similar to the baseline symmetrically-initiated scheme but the location policy is adaptive.

Mirchandaney *et al.* [98, 99] studied the performance of sender-, receiver- and symmetrically-initiated schemes using matrix-geometric solution techniques for homogeneous systems. They studied the effect of delays in transferring jobs and gathering remote state information on the performance of the algorithms.

Benmohammed-Mahieddine *et al.* [12] proposed a symmetrically-initiated scheme that uses periodic polling of a single random processor to acquire information about the system state. They designed this scheme to reduce the number of load balancing messages exchanged at heavy load levels under the baseline symmetrically-initiated scheme.

Leinberger *et al.* [87] studied the receiver-, sender- and symmetrically- initiated schemes in computational grids consisting of near-homogeneous multiresource servers.

There are other distributed dynamic load balancing schemes that cannot be classified using the three classes presented above. A large class of such schemes are the iterative schemes and schemes that use concepts from artificial intelligence.

- **Iterative load balancing schemes:**

  This is an important class of schemes in which the processors have to balance the load with their neighbors iteratively until the whole network is globally balanced [27]. There are two types of iterative schemes: *diffusion schemes* and *dimension exchange schemes*. In the diffusion schemes a processor in the network can send and receive messages to/from all its neighbors simultaneously whereas dimension exchange schemes use only pairwise communication iteratively balancing with one neighbor after the other [32, 42].

  Cybenko [32] proposed and studied a diffusion scheme in which tasks 'diffuse' from processors with excess tasks to neighboring processors that have fewer tasks. By examining the eigenstructure of the iteration matrix that arise from the model, the convergence properties of the algorithm are derived. Franklin and Govindan [49] extended the model of Cybenko such that it can represent several algorithms in addition to the diffusion scheme.

Elsasser *et al.* [42] generalized existing diffusion schemes in order to deal with heterogeneous processors. They studied the rate of convergence and the amount of load moved over the communication links. In general diffusion schemes exhibit slow convergence. To overcome this slow convergence Hu and Blake [61] proposed an improved diffusion scheme that uses global communication in addition to local communication.

Cortes *et al.* [28–30] proposed a diffusion scheme that detects unbalanced domains (a processor and its immediate neighbors) and correct this situation by allowing load movements between non-adjacent processors.

There exists several other studies of diffusion schemes that investigate: second order diffusion schemes [51], the running time of diffusion schemes [126] and the number of steps required to achieve load balancing [113].

- **Other distributed schemes:**

  Stankovic [124] proposed a heuristic load balancing scheme based on Bayesian Decision Theory. It was an initial attempt to use a distributed algorithm for load balancing under uncertainty. The scheme uses an adaptive observation procedure based on Bayesian evaluation techniques.

  There are other several approaches to load balancing that uses: neural networks [136], genetic algorithms [43, 85, 86], fuzzy decision algorithms [36], mobile agents [109, 121] and simulated annealing models [22].

There are no known pure game theoretic studies on dynamic load balancing problem, but there exists some related approaches derived from economic theory. These approaches are based on pricing, where consumers (tasks) and suppliers (processors) interact through a market mechanism which relies on resource prices and money. Ferguson *et al.* [47, 48] proposed a load balancing economy in a network of processors. Jobs entering the system purchase running and transmission time from processors. The prices for resources are determined via an auction mechanism. In [130], *Spawn* the first implementation of a distributed computational economy [62] is presented. *Spawn* addressed the problems of dynamic load balancing, resource contention and priority in distributed systems. The *Challenger* system presented in [23] is a multi-agent system that performs distributed load balancing using market based techniques. It consists of agents which manage local resources and cooperate in an attempt to optimize their utilization. All these works are based on auctioning games and thus they need to be mentioned in our survey of game theoretic approaches to load balancing in distributed systems.

2.2.3 Related problems

Routing traffic in networks is a closely related problem which was studied from a game theoretic perspective. Orda *et al*. [110] studied a noncooperative game in a network of parallel links with convex cost functions. They studied the existence and uniqueness of the Nash equilibrium. Altman *et al*. [6] investigated the same problem in a network of parallel links with linear cost functions. An important line of research was initiated by Koutsoupias and Papadimitriou [77], who considered a noncooperative routing game and proposed the *coordination ratio* (i.e. the ratio between the worst possible Nash equilibrium and the overall optimum) as a measure of effectiveness of the system. Mavronicolas and Spirakis [96] derived tight bounds on coordination ratio in the case of fully mixed strategies where each user assigns its traffic with non-zero probability to every link. Roughgarden and Tardos [119] showed that in a network in which the link cost functions are linear, the flow at Nash equilibrium has total latency at most 4/3 that of the overall optimal flow. They also showed that if the link cost functions are assumed to be only continuous and nondecreasing the total latency may be arbitrarily larger than the minimum possible total latency.

Korilis *et al.* [75] proposed a method for architecting noncooperative equilibria in the run time phase i.e. during the actual operation of the network. It requires the intervention of a manager that leads the system to an efficient Nash equilibrium. The manager does this by controlling its flow. Instead of reacting to the users, the manager fixes this optimal strategy and lets the users converge to their respective equilibrium.

This is a typical *Stackelberg game* (leader - follower game) in which the leader imposes its strategy on the self-optimizing users that behave as followers. Based on the work of Orda *et al.* [110] where the noncooperative routing problem is modeled as a static game, La and Anantharam [82] modeled the problem as a noncooperative repeated game. In a repeated game there is the possibility of strategies that result in Nash equilibria which are more efficient than in the single shot game. Such strategies are supported by credible threats or rewards that the user might make or offer to one another. The authors showed that in parallel link networks there always exists a Nash equilibrium that achieves the system-wide optimum cost and yet yields a cost for each user that is no greater than that of the unique stage game Nash equilibrium. They further proved that the Nash equilibrium is *subgame perfect* i.e. the strategies involved result in a Nash equilibrium in every subgame of the overall game.

Economides and Silvester [39,41] considered a noncooperative routing problem for two classes of packets. The first class objective is to minimize its *average packet delay*, while the other class objective is to minimize its *blocking probability*. They derived a routing policy for a two server system and presented the strategy and the performance of each class. They also studied in [40] the Nash equilibrium of the routing problem for two classes of users and two servers.

Game theory was also applied in other areas of modern networking such as: flow control [3–5,60,74,97,137], virtual bandwidth allocation [83] and pricing [26]. Recently, applications of game theory to computer science have attracted a lot of interest and have become a major trend. It is worth mentioning the recent DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design [1].

Chapter 3

A Cooperative Load Balancing Game

3.1 Introduction

Most of the previous work on static load balancing considered as their main objective the minimization of overall expected response time. The fairness of allocation, which is an important issue for modern distributed systems, has received relatively little attention. In this chapter we consider the load balancing problem in single class job distributed systems. Our goal is to find a formal framework for characterization of fair load balancing schemes that are also optimal for each job. The framework was provided by cooperative game theory. Using this framework we formulate the load balancing problem in single class job distributed systems as a cooperative game among computers. We show that the *Nash Bargaining Solution*(NBS) provides a Pareto optimal operation point for the distributed system. We give a characterization of NBS and an algorithm for computing it. We prove that the NBS is a fair solution and we compare its performance with other existing solutions. NBS guarantees the optimality and the fairness of allocation.

Organization

This chapter is structured as follows. In Section 3.2 we present the Nash Bargaining Solution concept for cooperative games and state several lemmas and theorems needed for our result. In Section 3.3 we introduce our cooperative load balancing game and derive an algorithm for computing the solution. In Section 3.4 the performance and fairness of our cooperative solution is compared with those of other existing solutions. In Section 3.5 we draw conclusions.

3.2 The Nash Bargaining Solution

In the following we discuss the *Nash Bargaining Solution (NBS)* [101,102] for cooperative games. The Nash Bargaining Solution is different from the Nash Equilibrium for noncooperative games. In a cooperative game the performance of each player may be made better than the performance achieved in a noncooperative game at the Nash Equilibrium.

Definition 3.1 (A cooperative game) A cooperative game consists of:

- M *players*;
- A nonempty, closed and convex set $X \subseteq \mathbf{R}^M$ which is the *set of strategies* of the M players.
- For each player i, $i = 1, 2, \ldots, M$, an *objective function* f_i. Each f_i is a function from X to $\mathbf{R}$ and it is bounded above. The goal is to maximize simultaneously all $f_i(x)$.
- For each player i, $i = 1, 2, \ldots, M$, a *minimal value* of f_i, denoted u_i^0, required by player i without any cooperation to enter the game. The vector $\mathbf{u}^0 = (u_1^0, u_2^0, \ldots, u_M^0)$ is called the initial *agreement point.*

Example 3.1 (A cooperative game in distributed systems) In the context of resource allocation in distributed systems the players could represent computers. The objective function $f_i(x)$ could represent the inverse of the expected response time at computer i, where x is the vector of job arrival rates at each computer. The initial values u_i^0 of each computer represent a minimum performance guarantee the computers must provide to the users.

Definition 3.2 (The set of achievable performances) The *set of achievable performances with respect to the initial agreement point* $\mathbf{u}^0$ is the set G defined by: $G = \{(U, \mathbf{u}^0) \mid U \subset \mathbf{R}^M$ is a nonempty convex and bounded set, and $\mathbf{u}^0 \in \mathbf{R}^M$ is such that $U_0 = \{\mathbf{u} \in U \mid \mathbf{u} \geq \mathbf{u}^0\}$ is nonempty $\}$ where $U \subset \mathbf{R}^M$ is the set of achievable performances.

We are interested in finding solutions for the cooperative game defined above that are Pareto optimal. In the following we define the notion of Pareto optimality.

Definition 3.3 (Pareto optimality) Assume $x \in X$ and $\mathbf{f}(x) = (f_1(x), \ldots, f_M(x))$, $\mathbf{f}(x) \in U$. Then x is said to be *Pareto optimal* if for each $x^{'} \in X$, $f_i(x^{'}) \geq f_i(x)$, $i = 1, \ldots, M$ imply $f_i(x^{'}) = f_i(x)$, $i = 1, \ldots, M$.

Remark: Pareto optimality means that it is impossible to find another point which leads to strictly superior performance for all the players.

In general, for an M-player game the set of Pareto optimal points form an $M - 1$ dimensional hypersurface consisting of an infinite number of points [112]. What is the desired operating point for our system among them? To answer this question we need additional criteria for selecting the optimal point. Such criteria are the so called fairness axioms that characterize the *Nash Bargaining Solution*.

Definition 3.4 (The Nash Bargaining Solution (NBS)) [125] A mapping $S : G \rightarrow \mathbf{R}$ is said to be a *Nash Bargaining Solution* if:

i) $S(U, \mathbf{u}^0) \in U_0$;

ii) $S(U, \mathbf{u}^0)$ is Pareto optimal;

and satisfies the following axioms:

iii) *Linearity axiom:* If $\phi : \mathbf{R}^M \rightarrow \mathbf{R}^M$, $\phi(\mathbf{u}) = \mathbf{u}^{'}$ with $u_j^{'} = a_j u_j + b_j$, $a_j > 0$, $j = 1, \ldots, k$ then $S(\phi(U), \phi(\mathbf{u}^0)) = \phi(S(U, \mathbf{u}^0))$.

iv) *Irrelevant alternatives axiom:* If $U \subset U^{'}$, $(U, \mathbf{u}^0) \in G$ and $S(U^{'}, \mathbf{u}^0) \in U$, then $S(U, \mathbf{u}^0) = S(U^{'}, \mathbf{u}^0)$.

v) *Symmetry axiom:* If U is symmetrical with respect to a subset $J \subseteq \{1, \ldots, M\}$ of indices (i.e. if $\mathbf{u} \in U$ and $i, j \in J$, $i < j$ imply $(u_1, \ldots, u_{i-1}, u_j, u_{j+1}, \ldots, u_{j-1}, u_i, u_{j+1}, \ldots, u_M) \in U$) and if $u_i^0 = u_j^0$, $i, j \in J$ then $S(U, \mathbf{u}^0)_i = S(U, \mathbf{u}^0)_j$, for $i, j \in J$.

Definition 3.5 (Bargaining point) $\mathbf{u}^*$ is a *bargaining point* if it is given by $S(U, \mathbf{u}^0)$. We call $\mathbf{f}^{-1}(\mathbf{u}^*)$ the set of *bargaining solutions*.

Remarks: Axioms iii)-v) are called the *fairness axioms*. Essentially they say the following. The NBS is unchanged if the performance objectives are affinely scaled (Axiom iii). The bargaining point is not affected by enlarging the domain (Axiom iv). The bargaining point does not depend on the specific labels, i.e. players with the same initial points and objectives will obtain the same performance (Axiom v).

Stefanescu [125] gave the following characterization of the Nash bargaining point.

Theorem 3.1 (Nash bargaining point characterization) Let X be a convex compact subset of $\mathbf{R}^M$. Let $f_i : X \to \mathbf{R}$, $i = 1, \ldots, M$ be concave functions, bounded above. Let $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}))$, $U = \{\mathbf{u} \in \mathbf{R}^M \mid \exists \mathbf{x} \in X, \mathbf{f}(\mathbf{x}) \geq \mathbf{u}\}$, $X(\mathbf{u}) = \{\mathbf{x} \in X \mid \mathbf{f}(\mathbf{x}) \geq \mathbf{u}\}$ and $X_0 = X(\mathbf{u}^0)$ be the set of strategies that enable the players to achieve at least their initial performances. Then:

(i) There exists a bargaining solution and a unique bargaining point $\mathbf{u}^*$.

(ii) The set of the bargaining solutions $\mathbf{f}^{-1}(\mathbf{u}^*)$ is determined as follows:

Let J be the set of players able to achieve a performance strictly superior to their initial performance, i.e. $J = \{j \in \{1, \ldots, M\} \mid \exists \mathbf{x} \in X_0, f_j(\mathbf{x}) > u_j^0\}$. Each vector $\mathbf{x}$ in the bargaining solution set verifies $f_j(\mathbf{x}) > u_j^0$, for all $j \in J$ and solves the following maximization problem:

$$\max_{\mathbf{x}} \prod_{j \in J} (f_j(\mathbf{x}) - u_j^0) \qquad \mathbf{x} \in X_0 \tag{3.1}$$

Hence $\mathbf{u}^*$ satisfies $u_j^* > u_j^0$ for $j \in J$ and $u_j^* = u_j^0$ otherwise.

Remark: From the assumption on J, the product in equation (3.1) is positive. The players outside of J are not considered in the optimization problem.

Yaiche *et al.* [135] formulated an equivalent optimization problem and proved the following results. These results are needed for proving Theorem 3.4 and 3.5.

Theorem 3.2 If each $f_j : X \to \mathbf{R}$, ($j \in J$) is one-to-one on $X_0 \subset X$ then $\mathbf{f}^{-1}(\mathbf{u}^*)$ is a singleton.

Remark: Following the terminology used in [135], we call the point in this set the Nash Bargaining Solution in the rest of the paper.

Lemma 3.1 Let $g : X \to \mathbf{R}_+ \backslash \{0\}$ be a concave function, where $\mathbf{R}_+$ is the set of nonnegative real numbers. Then $h = \ln(g(.)) : X \to \mathbf{R}$ is concave. If g is one-to-one, then h is strictly concave.

Theorem 3.3 Suppose that for each $j \in J$, $f_j : X \to \mathbf{R}$, is one-to-one on X_0. Under the assumption of Theorem 3.1, we consider the following problems:

$$(P_J) : \qquad \max_{\mathbf{x}} \prod_{j \in J} (f_j(\mathbf{x}) - u_j^0) \qquad \mathbf{x} \in X_0 \tag{3.2}$$

$$(P_J') : \qquad \max_{\mathbf{x}} \sum_{j \in J} \ln(f_j(\mathbf{x}) - u_j^0) \qquad \mathbf{x} \in X_0 \tag{3.3}$$

then:

1. (P_J) has a unique solution and the bargaining solution is a single point.
2. (P_J') is a convex optimization problem and has a unique solution.
3. (P_J) and (P_J') are equivalent. The unique solution of (P_J') is the bargaining solution.

Yaiche *et al.* [135] used this equivalent problem and derived a game theoretic model for bandwidth allocation and pricing in broadband networks.

3.3 Load balancing as a cooperative game among computers

We consider a single class job distributed system that consists of n heterogeneous computers. We consider a game in which each computer is a player and it must minimize the expected execution time of jobs that it processes. If $F_i(\beta_i)$ denotes the expected execution time of jobs processed at computer i, we can express the game as follows:

$$\min_{\beta_i} F_i(\beta_i), \qquad i = 1, \dots, n \tag{3.4}$$

where β_i is the average arrival rate of jobs at computer i.

Modeling each computer as an M/M/1 queueing system [94], $F_i(\beta_i) = \frac{1}{\mu_i - \beta_i}$, where μ_i is the average service rate of computer i. The minimization problem, associated with the above game, becomes:

$$\min_{\beta_i} \frac{1}{\mu_i - \beta_i}, \qquad i = 1, \dots, n \tag{3.5}$$

subject to:

$$\beta_i < \mu_i, \qquad i = 1, \dots, n \tag{3.6}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.7}$$

$$\beta_i \geq 0, \qquad i = 1, \dots, n \tag{3.8}$$

where Φ is the total job arrival rate of the system. The first constraint is the ‘stability’ condition and the second one is the ‘conservation law’ for the M/M/1 system.

This problem is equivalent to:

$$\max_{\beta_i}(-\beta_i), \qquad i = 1, \dots, n \tag{3.9}$$

subject to:

$$-\beta_i > -\mu_i, \qquad i = 1, \dots, n \tag{3.10}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.11}$$

$$-\beta_i \leq 0, \qquad i = 1, \dots, n \tag{3.12}$$

Based on this optimization problem we can formulate an equivalent game as follows.

Definition 3.6 (The cooperative load balancing game) The cooperative load balancing game consists of:

- n computers as *players*;
- The *set of strategies*, X, is defined by the following constraints:

$$-\beta_i \geq -\mu_i, \qquad i = 1, \dots, n \tag{3.13}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.14}$$

$$-\beta_i \leq 0, \qquad i = 1, \dots, n \tag{3.15}$$

- For each computer i, $i = 1, 2, \ldots, n$, the *objective function* $f_i(\beta_i) = -\beta_i$, $\beta_i \in X$. The goal is to maximize simultaneously all $f_i(\beta_i)$.

- For each computer i, $i = 1, 2, \ldots, n$, the initial performance $u_i^0 = -\mu_i$. This is the value of f_i required by computer i without any cooperation to enter the game.

Remarks:

(i) To satisfy the compactness requirement for X we allow the possibility that $\beta_i = \mu_i$. This requirement will be dropped in the following.

(ii) We assume that all n computers in the set J of players are able to achieve performance strictly superior to the initial performance. We assumed that the initial agreement point is $u_i^0 = -\mu_i$, $i = 1, \ldots, n$. In other words all computers agree that they will choose β_i such that $-\beta_i > -\mu_i$. This is also the 'stability' condition for the M/M/1 system.

Now we are interested in finding the Nash Bargaining Solution for the game defined above. Our results are given in Theorem 3.4 - 3.8. Their proofs are presented in the Appendix A.

Theorem 3.4 For the load balancing cooperative game defined above there is a unique bargaining point and the bargaining solutions are determined by solving the following optimization problem:

$$\max_{\beta} \prod_{i=1}^{n} (\mu_i - \beta_i) \tag{3.16}$$

subject to:

$$\beta_i < \mu_i, \qquad i = 1, \ldots, n \tag{3.17}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.18}$$

$$\beta_i \geq 0, \qquad i = 1, \ldots, n \tag{3.19}$$

Theorem 3.5 For the load balancing cooperative game defined above the bargaining solution is determined by solving the following optimization problem:

$$\max_{\beta} \sum_{i=1}^{n} \ln(\mu_i - \beta_i) \tag{3.20}$$

subject to:

$$\beta_i < \mu_i, \qquad i = 1, \ldots, n \tag{3.21}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.22}$$

$$\beta_i \geq 0, \qquad i = 1, \ldots, n \tag{3.23}$$

As a first step in obtaining the solution for our load balancing cooperative game we solve the optimization problem given in Theorem 3.5 without requiring β_i ($i = 1, \ldots, n$) be non-negative. The solution of this problem is given in the following theorem.

Theorem 3.6 The solution of the optimization problem in Theorem 3.5 without the constraint $\beta_i \geq 0$, $i = 1, \ldots, n$ is given by:

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n} \tag{3.24}$$

In practice we cannot use this solution because there is no guarantee that β_i ($i = 1, \ldots, n$) is always non-negative. Note that β_k is negative when $\mu_k < \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n}$. This means that computer k is very slow. In such cases we make the solution feasible by setting $\beta_k = 0$ and remove computer k from the system. Setting β_k equal to zero means that we do not assign jobs to the extremely slow computers. Assuming that computers are ordered in decreasing order of their processing rates, we eliminate the slowest computer and recompute the allocation for a system with $n - 1$ computers. This procedure is applied until a feasible solution is found.

Based on this fact and Theorem 3.6, we derive a centralized algorithm (called COOP) for obtaining the Nash Bargaining Solution for the load balancing cooperative game. In the following we present this algorithm:

COOP algorithm:

```
Input: Average processing rates: $\mu_1, \mu_2, \dots \mu_n$; Total arrival rate: $\Phi$
Output: Load allocation: $\beta_1, \beta_2, \dots \beta_n$;
    1. Sort the computers in decreasing order of their average
           processing rate ($\mu_1 \geq \mu_2 \geq \dots \geq \mu_n$);
    2. $c \leftarrow \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n}$
    3. while ( $c > \mu_n$ ) do
           $\beta_n \leftarrow 0$
           $n \leftarrow n - 1$
           $c \leftarrow (c - \frac{\mu_{n+1}}{n+1}) \frac{n+1}{n}$
    4. for $i = 1, \dots, n$ do
           $\beta_i \leftarrow \mu_i - c$
```

The following theorem proves the correctness of this algorithm.

Theorem 3.7 (Correctness) The allocation $\{\beta_1, \beta_2, \dots, \beta_n\}$ computed by the COOP algorithm solves the optimization problem in Theorem 3.5 and is the NBS for our cooperative load balancing game.

The execution time of this algorithm is in $O(n \log n)$. In general determining the NBS is an NP-hard problem [35]. In our case we were able to obtain an $O(n \log n)$ algorithm because the cooperative load balancing game is a convex game (i.e. player's objective functions are convex).

Example 3.2 (Applying COOP) We consider a distributed system consisting of 3 computers with the following processing rates: $\mu_1 = 8.0$ jobs/sec, $\mu_2 = 5.0$ jobs/sec and $\mu_3 = 2.0$ jobs/sec. The total arrival rate is $\Phi = 7.0$ jobs/sec. The computers are already sorted in decreasing order of their processing rates and we execute Step 2 in which we compute c.

$$c = \frac{15.0 - 7.0}{3} = \frac{8}{3}$$

The while loop in Step 3 is executed because $c > \mu_3$. In this loop $\beta_3 = 0$, $n = 2$ and c is updated to 3 and then the algorithm proceeds to Step 4. In this step the values of the loads are computed: $\beta_1 = 8.0 - 3.0 = 5.0$ jobs/sec and $\beta_2 = 5.0 - 3.0 = 2.0$ jobs/sec.

The *fairness index* (defined from the jobs' perspective),

$$I(\mathbf{T}) = \frac{[\sum_{i=1}^{n} T_i]^2}{n \sum_{i=1}^{n} T_i^2} \tag{3.25}$$

was proposed in [65] to quantify the fairness of load balancing schemes. Here the parameter $\mathbf{T}$ is the vector $\mathbf{T} = (T_1, T_2, \ldots, T_n)$ where T_i is the average execution time for jobs that are processed at computer i.

Remark: This index is a measure of the 'equality' of execution times at different computers. So it is a measure of load balance. If all the computers have the same expected job execution times then $I = 1$ and the system is 100% fair to all jobs and is load balanced. If the differences on T_i increase, I decreases and the load balancing scheme favors only some tasks.

For the proposed load balancing cooperative game we can state the following:

Theorem 3.8 (Fairness) The fairness index equals 1 when we use the COOP algorithm to solve the proposed load balancing cooperative game.

This means that all jobs receive a fair treatment independent of the allocated computer.

3.4 Experimental Results

3.4.1 Simulation Environment

The simulations were carried out using Sim++ [31], a simulation software package written in C++. This package provides an application programming interface which allow the programmer to call several functions related to event scheduling, queueing, preemption and random number generation. The simulation model consists of a collection of computers connected by a communication network. Jobs arriving at the system are distributed by a central dispatcher to the computers according to the specified load balancing scheme. Jobs which have been dispatched to a particular computer are *run-to-completion* (i.e. no preemption) in FCFS (first-come-first-served) order.

Each computer is modeled as an M/M/1 queueing system [73]. The main performance metrics used in our simulations are the *expected response time* and the *fairness index*. The simulations were run over several millions of seconds, sufficient to generate a total of 1 to 2 millions jobs typically. Each run was replicated five times with different random number streams and the results averaged over replications. The standard error is less than 5% at the 95% confidence level.

3.4.2 Performance Evaluation

For comparison purposes we consider three static load balancing schemes. In addition to our cooperative static scheme (COOP) we implemented three static schemes. A brief description of these schemes is given below:

- **Proportional Scheme (PROP)** [24]: According to this scheme jobs are allocated in proportion to the processing speed of computers. The following centralized algorithm is used for obtaining the load allocation.

  PROP algorithm:

  ```
  Input: Average processing rates: $\mu_1, \mu_2, \dots \mu_n$; Total arrival rate: $\Phi$
  Output: Load allocation: $\beta_1, \beta_2, \dots \beta_n$;
      for $i = 1, \dots, n$ do
          $\beta_i \leftarrow \Phi \frac{\mu_i}{\sum_{j=1}^{n} \mu_j}$
  ```

  This allocation seems to be a natural choice but it may not minimize the average response time of the system and is unfair. The running time of PROP is $O(n)$.

- **Overall Optimal Scheme (OPTIM)** [127, 128]: This scheme minimizes the expected execution time over all jobs executed by the system. The loads at each computer (β_i) are obtained by solving the following nonlinear optimization problem:

  $$\min \frac{1}{\Phi} \sum_{i=1}^{n} \beta_i F_i(\beta_i) \tag{3.26}$$

  subject to the constraints:

  $$\beta_i < \mu_i, \qquad i = 1, \dots, n \tag{3.27}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{3.28}$$

$$\beta_i \geq 0, \qquad i = 1, \ldots, n \tag{3.29}$$

The following centralized algorithm is used for obtaining the load allocation.

OPTIM algorithm:

```
Input: Average processing rates: $\mu_1$, $\mu_2$, ... $\mu_n$; Total arrival rate: $\Phi$
Output: Load allocation: $\beta_1$, $\beta_2$, ... $\beta_n$;
    1. Sort the computers in decreasing order of their average
           processing rate ($\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$);
    2. $c \leftarrow \frac{\sum_{i=1}^{n} \mu_i - \Phi}{\sum_{i=1}^{n} \sqrt{\mu_i}}$
    3. while ( $c > \sqrt{\mu_n}$ ) do
           $\beta_n \leftarrow 0$
           $n \leftarrow n - 1$
           $c \leftarrow \frac{\sum_{i=1}^{n} \mu_i - \Phi}{\sum_{i=1}^{n} \sqrt{\mu_i}}$
    4. for $i = 1, \ldots, n$ do
           $\beta_i \leftarrow \mu_i - c\sqrt{\mu_i}$
```

This scheme provides the *overall optimum* (global approach) for the expected execution time but is unfair. The running time of OPTIM is $O(nlogn)$, (due to the sorting in step 1).

- **Wardrop Equilibrium Scheme (WARDROP)** [67]: In this scheme each of infinitely many jobs optimizes its response time for itself independently of others (noncooperative approach). In general the Wardrop equilibrium solution is not Pareto optimal and in some cases we expect worse response time than the other policies [67]. The algorithm for obtaining the Wardrop equilibrium is a centralized algorithm based on an iterative procedure that is not very efficient. For a complete description of WARDROP algorithm see [67]. The advantage of this scheme is that it provides a fair allocation. The running time of WARDROP is $O(nlogn + nlog(1/\epsilon))$, where ϵ denotes the acceptable tolerance used in computing the allocation [67].

Remark: Among the three schemes described above, the WARDROP scheme is the only scheme that is based on game theoretic concepts.

We evaluated the schemes presented above under various system loads and configurations. In the following we present and discuss the simulation results.

Effect of System Utilization

To study the effect of system utilization we simulated a heterogeneous system consisting of 16 computers with four different processing rates. In Table 3.1 we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We choose 0.013 jobs/sec. as the the processing rate for the slowest computer because it is a value that can be found in real distributed systems [127]. Also we consider only computers that are at most ten times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

Table 3.1. System configuration.

Relative processing rate	1	2	5	10
Number of computers	6	5	3	2
Processing rate (jobs/sec)	0.013	0.026	0.065	0.13

In Figure 3.1 we present the mean response time of the system for different values of system utilization (ranging from 10% to 90%). *System utilization* (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i} \tag{3.30}$$

It can be observed that at low loads (ρ from 10% to 40%) all the schemes except PROP yield almost the same performance. The poor performance of PROP scheme is due to the fact that the less powerful computers are significantly overloaded.

At medium loads (ρ from 40% to 60%) the COOP scheme performs significantly better than PROP and approaches the performance of OPTIM. For example at load level of 50% the expected response time of

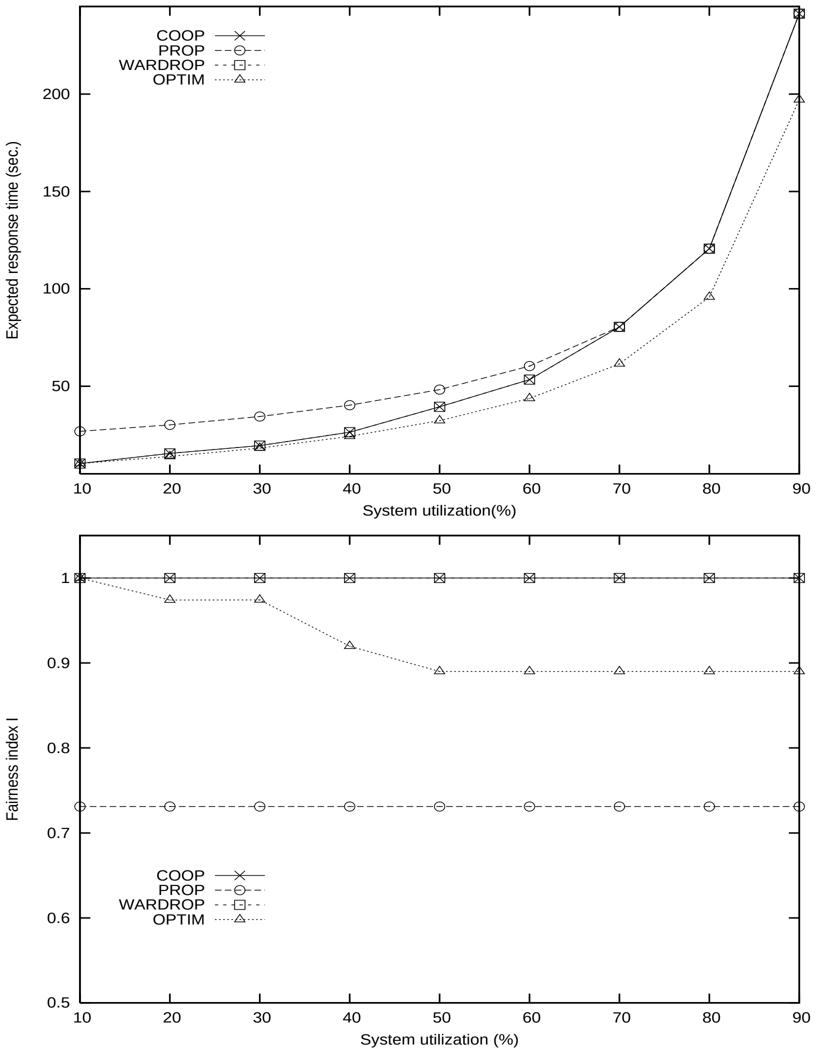

Figure 3.1. The expected response time and fairness index vs. system utilization.

COOP is 19% less than PROP and 20% greater than OPTIM. At high loads COOP and PROP yield the same expected response time which is greater than that of OPTIM. The WARDROP and COOP yield the same performance for the whole range of system utilization. The reason for this is that the noncooperative game used to derive the Wardrop equilibrium is a convex game and it has a unique equilibrium that is a Pareto optimal solution [69].

An important performance metric is the fairness index. The COOP and WARDROP schemes maintain a fairness index of 1 over the whole range of system loads and they are the only fair schemes here. It can be shown that for the distributed system considered in our simulations PROP has a fairness index of 0.731

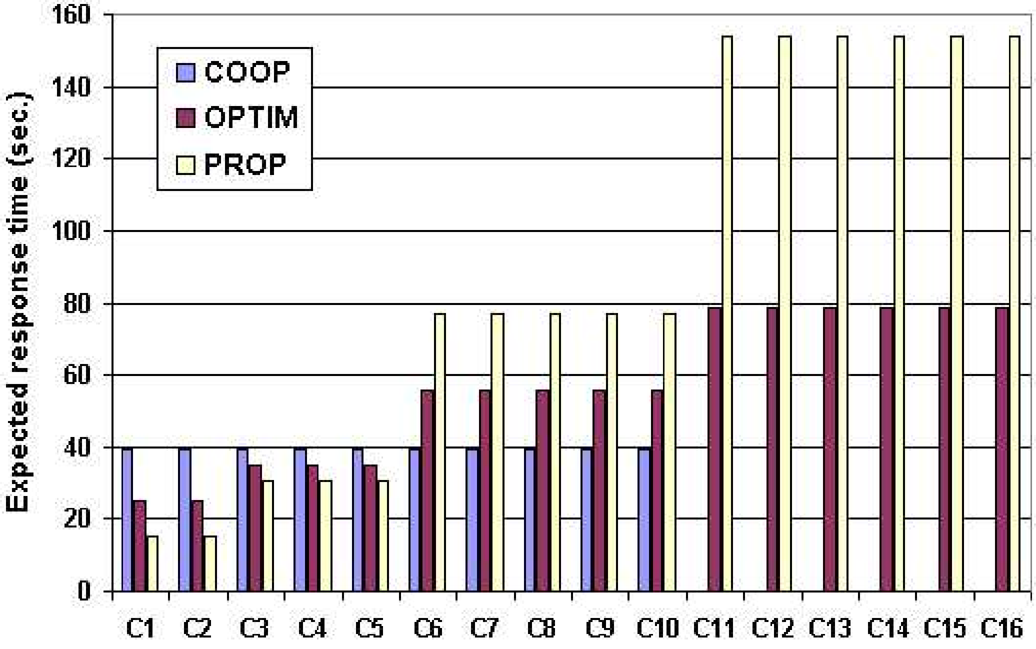

Figure 3.2. Expected response time at each computer (medium system load).

which is a constant independent of the system load. The fairness index of OPTIM varies from 1 at low load, to 0.88 at high load.

An interesting issue is the impact of static load balancing schemes on individual computers. In Figure 3.2 we present the expected response time at each computer for all static schemes at medium load (ρ=50%). The WARDROP scheme gives the same results as COOP and is not presented in these figures. Our scheme (COOP) guarantees equal expected response times for all computers. The value of the expected response time for each computer is equal to the value of overall expected response time (39.44 sec). This means that jobs would have the same expected response time independent of the allocated computers. We can observe that some of the slowest computers are not utilized by the COOP scheme (C_{11} to C_{16}). The PROP scheme overloads the slowest computers and the overall expected response time is increased. The difference in the expected execution time at C_1 (fastest) and C_{16} (slowest) is significant, 15 sec compared with 155 sec. In the case of OPTIM the expected response times are more balanced and the slowest computers are not overloaded.

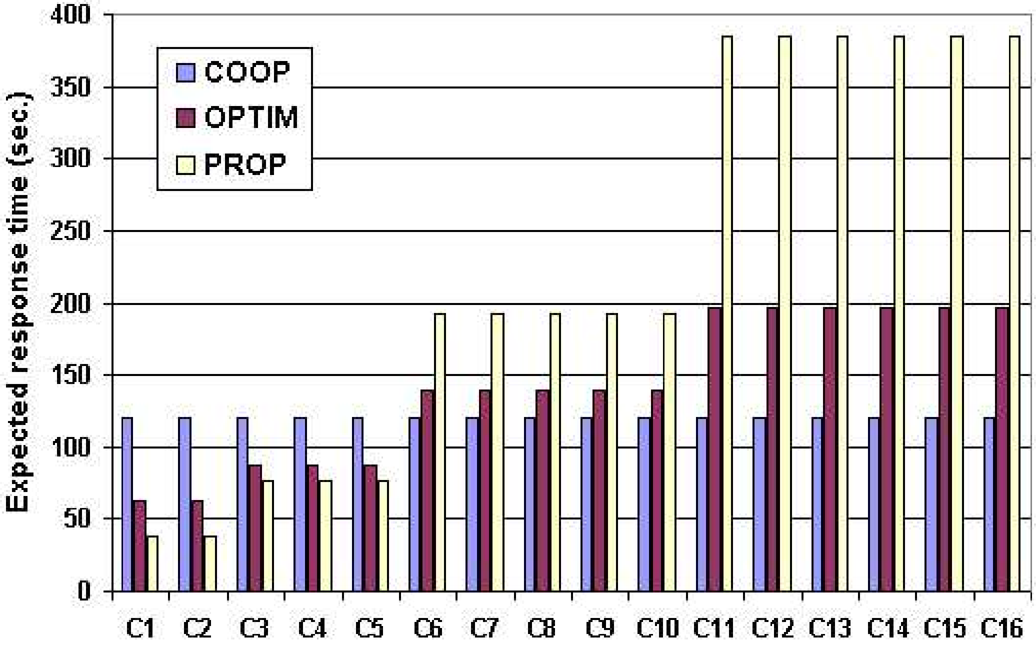

Figure 3.3. Expected response time at each computer (high system load).

In the case of PROP and OPTIM jobs are treated unfairly in the sense that a job allocated to a fast computer will have a low expected response time and a job allocated to a slow computer will have a high expected response time.

In Figure 3.3 we present the expected response time at each computer in the system when the system is heavily loaded ($\rho = 80\%$). The COOP scheme guarantees the same execution time at each computer and utilizes all the computers. The difference in the expected response time between the fastest and slowest computers is huge in the case of PROP (350 sec.) and moderate in the case of OPTIM (130 sec.). COOP scheme provides really a fair and load balanced allocation and in some systems this is a desirable property.

Remark: **(COOP vs. WARDROP)** The complexity of WARDROP is higher than that of COOP. The reason is that in the COOP scheme the algorithm for computing the allocation is very simple. The WARDROP scheme involves a more complicated algorithm that require an iterative process that takes more time to converge. The algorithmic complexity of WARDROP is $O(nlogn + nlog(1/\epsilon))$, where ϵ denotes the acceptable tolerance used in computing the allocation. The complexity of COOP is $O(nlogn)$. The hidden constants

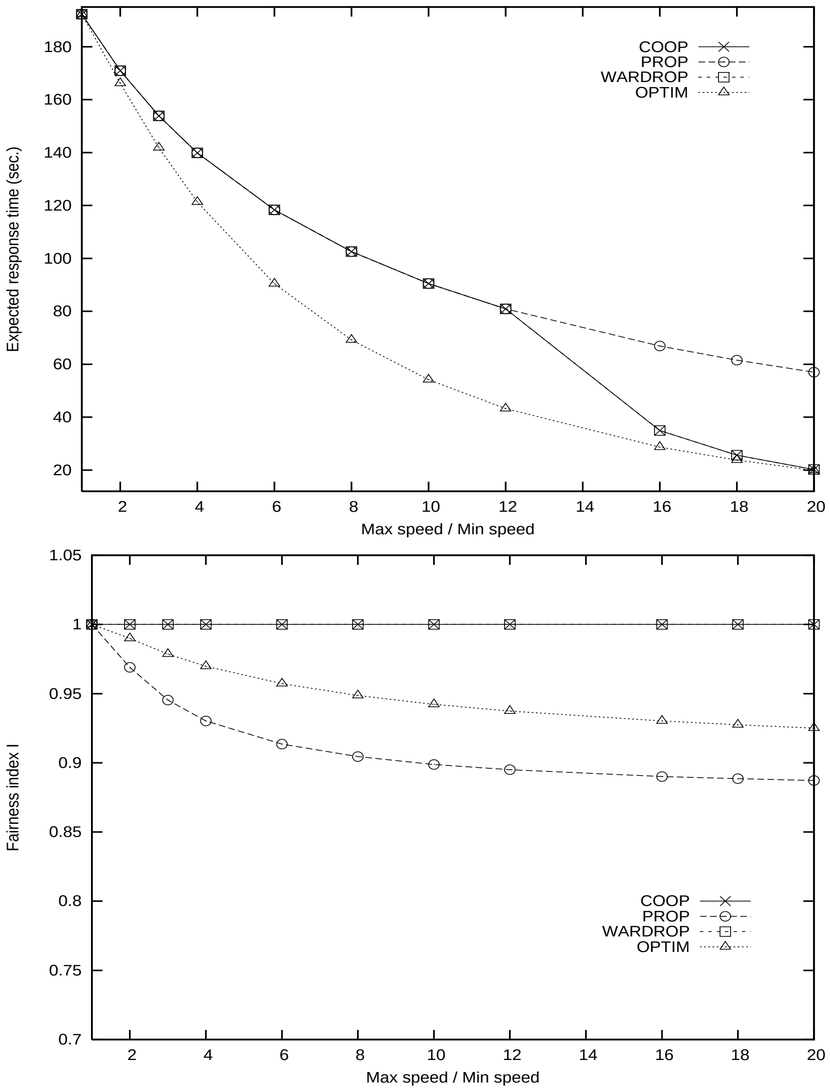

Figure 3.4. The effect of heterogeneity on the expected response time and fairness index.

are much higher in the case of WARDROP due to the high number of operations executed at each step. For common values of ϵ (from 10^{-5} to 10^{-7}) and n (from 10 to 100) the execution time of WARDROP is much higher because $1/\epsilon \gg n$. As an example, we ran both algorithms for a system of 16 computers on a SUN 10 (440 MHz) workstation and we obtained the following execution times: 4.1 msec for WARDROP ($\epsilon = 10^{-5}$) and 0.8 msec for COOP.

Effect of heterogeneity

In a distributed system, heterogeneity usually consists of: processor speed, memory and I/O. A simple way to characterize system heterogeneity is to use the processor speed. Furthermore it is reasonable to assume that a computer with high speed processor will have matching resources (memory and I/O). One of the common measures of heterogeneity is the *speed skewness* which is defined as the ratio of maximum processing rate to minimum processing rate of the computers. This measure is somehow limited but for our goals it is satisfactory.

In this section we investigate the effectiveness of load balancing schemes by varying the speed skewness. We simulate a system of 16 heterogeneous computers: 2 fast and 14 slow. The slow computers have a relative processing rate of 1 and we varied the relative processing rate of the fast computers from 1 (which correspond to a homogeneous system) to 20 (which correspond to a highly heterogeneous system). The system utilization was kept constant, ρ = 60%.

In Figure 3.4 we present the effect of speed skewness on the expected response time and fairness. It can be observed that increasing the speed skewness the OPTIM and COOP schemes yield low response times which means that in highly heterogeneous systems the COOP and OPTIM are very effective. COOP has the additional advantage of a fair allocation which is very important in some systems. PROP scheme performs poorly because it overloads the slowest computers.

Effect of system size

An important issue is to study the influence of system size on the performance of load balancing schemes. To study this issue we simulated a heterogeneous distributed system consisting of two types of computers: slow computers (relative processing rate = 1) and fast computers (relative processing rate = 10). Figure 3.5 shows the expected response time and the fairness index where the number of computers increases from 2 (fast computers only) to 20 (2 fast and 18 slow computers). The performance of OPTIM and COOP is almost the same when we have few computers (2 to 8). The PROP scheme performs poorly even for a small system. The expected response time for COOP degrades increasing the system size and approaches the expected response time of PROP. COOP guarantees a good performance for small systems and the same

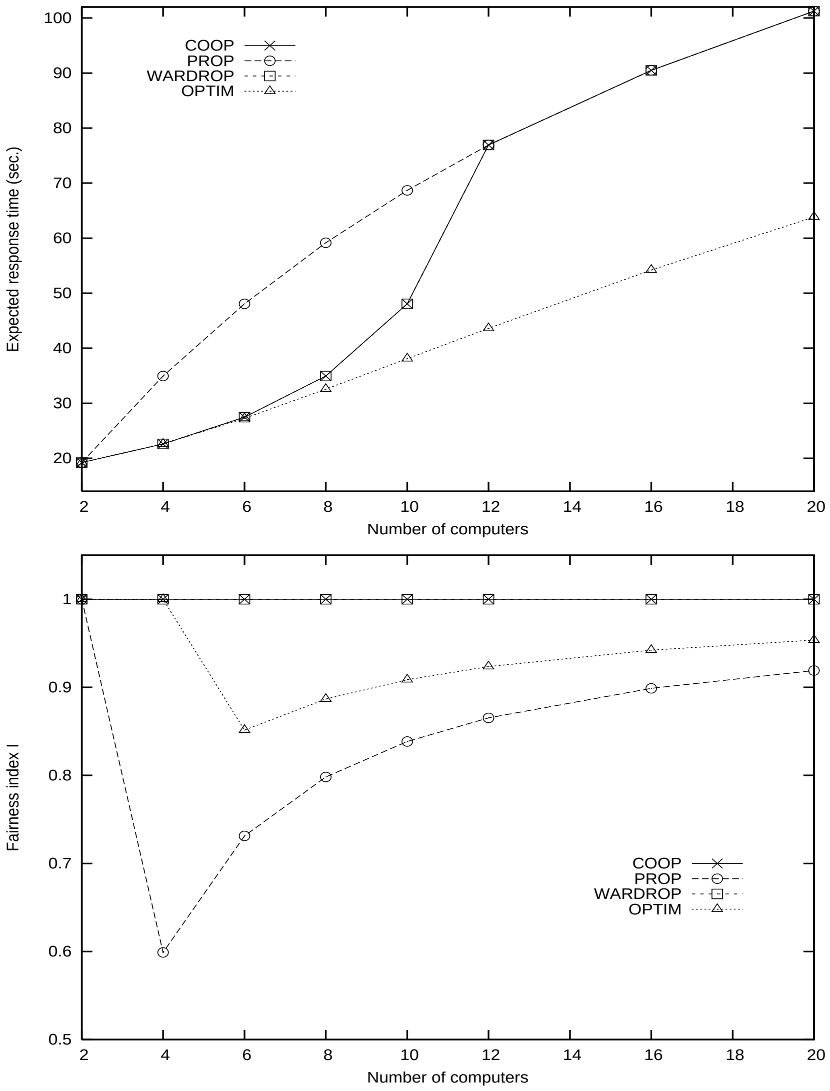

Figure 3.5. The effect of system size on the expected response time and fairness.

performance as PROP but with the advantage of a fair allocation in the case of large systems.

Experiments using the hyper-exponential distribution for the inter-arrival times

We investigate the performance of our load balancing scheme by considering a job arrival process different from the Poisson distribution. We use the two-stage hyper-exponential distribution for modeling the job arrival process [73]. The parameters of the system used in the simulation are those presented in Table 3.1. The coefficient of variation for the job inter-arrival time is set to 1.6.

In Figure 3.6, we present the expected response time of the system for different values of system utiliza-

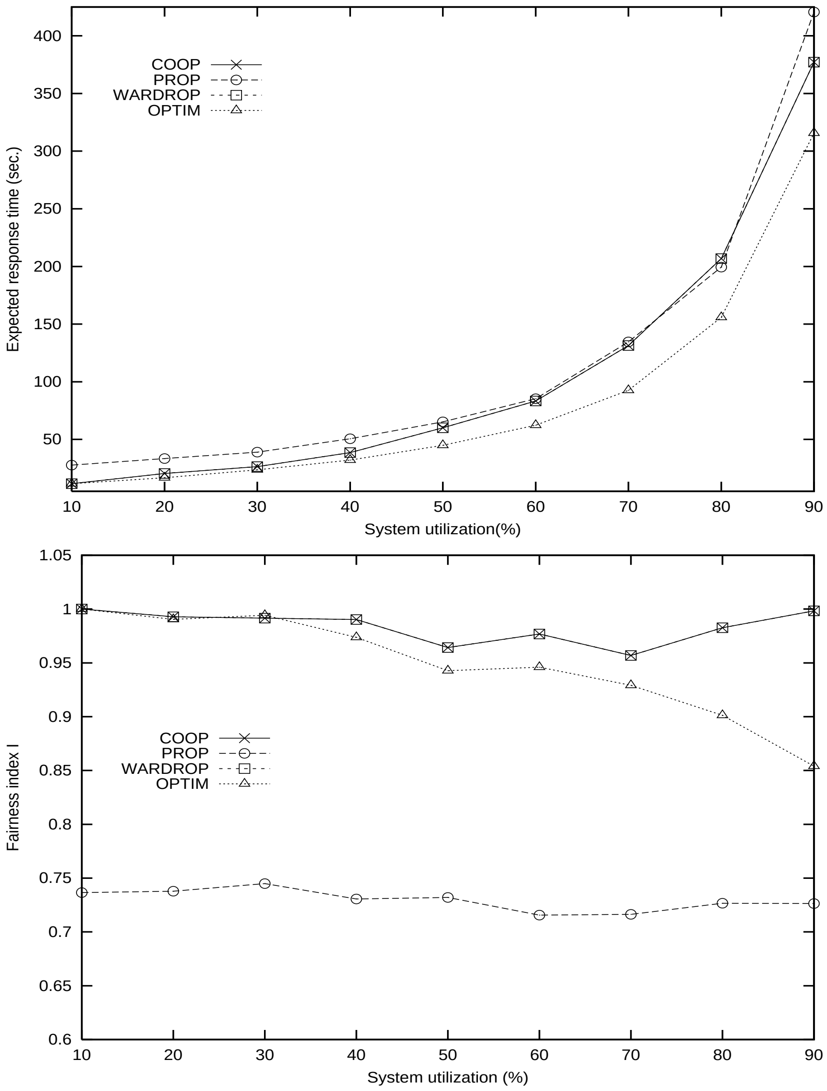

Figure 3.6. Expected response time and fairness (hyper-exponential distribution of arrivals).

tion (ranging from 10% to 90%). It can be observed that the performance is similar to that obtained using the Poisson distribution for the arrival process. At low loads all the schemes except PROP yield almost the same performance. At medium loads (ρ from 40% to 60%) the COOP scheme performs significantly better than PROP and approaches the performance of OPTIM. At load level of 50% the expected response time of COOP is 8% lower than PROP and 16% higher than OPTIM. At load level of 90% the expected response time of COOP is 11% lower than PROP and 17% higher than OPTIM. The WARDROP and COOP yield the same performance for the whole range of system utilization. The COOP and WARDROP schemes maintain a fairness index between 0.95 and 1 over the whole range of system loads. The fairness index of OPTIM varies from 1 at low loads, to 0.85 at high loads.

3.5 Conclusion

In this chapter we have presented a game theoretic framework for obtaining a fair load balancing scheme. The main goal was to derive a fair and optimal allocation scheme. We formulated the load balancing problem in single class job distributed systems as a cooperative game among computers. We showed that the Nash Bargaining Solution (NBS) of this game provides a Pareto optimal operation point for the distributed system and it is also a fair solution. For the proposed cooperative load balancing game we presented the structure of the NBS. Based on this structure we derived a new algorithm for computing it. We showed that the fairness index is always 1 using our new cooperative load balancing scheme, which means that the solution is fair to all jobs. We compared the performance of our cooperative load balancing scheme with other existing schemes. The main advantages of our cooperative scheme are the simplicity of the underlying algorithm and the fair treatment of all jobs independent of the allocated processors.

Chapter 4

A Noncooperative Load Balancing Game

4.1 Introduction

The main objective of the static load balancing is the minimization of overall expected response time. This is difficult to achieve in distributed systems where there is no central authority controlling the allocation and users are free to act in a selfish manner. Our goal is to find a formal framework for characterizing user-optimal allocation schemes in distributed systems. The framework was provided by noncooperative game theory which has been applied to routing and flow control problems in networks but not to load balancing in distributed systems. Using this framework we formulate the load balancing problem in distributed systems as a noncooperative game among users. The Nash equilibrium provides a user-optimal operation point for the distributed system. We give a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compare the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system.

Organization

This chapter is structured as follows. In Section 4.2 we present the system model and we introduce our load balancing noncooperative game. In Section 4.3 we derive a greedy distributed algorithm for computing the Nash equilibrium for our load balancing game. In Section 4.4 the performance of our load balancing scheme is compared with those of other existing schemes. In Section 4.5 we draw conclusions.

4.2 Load balancing as a noncooperative game among users

We consider a distributed system that consists of n heterogeneous computers shared by m users. Each computer is modeled as an M/M/1 queueing system (i.e. Poisson arrivals and exponentially distributed processing times) [73] and is characterized by its average processing rate μ_i, $i = 1, \ldots, n$. Jobs are generated by user j with an average rate ϕ_j, and $\Phi = \sum_{j=1}^{m} \phi_j$ is the total job arrival rate in the system. The total job arrival rate Φ must be less than the aggregate processing rate of the system (i.e. $\Phi < \sum_{i=1}^{n} \mu_i$). The system model is presented in Figure 4.1. The users have to decide on how to distribute their jobs to computers such that they will operate optimally. Thus user j ($j = 1, \ldots, m$) must find the fraction s_{ji} of all its jobs that are assigned to computer i ($\sum_{i=1}^{n} s_{ji} = 1$ and $0 \leq s_{ji} \leq 1$, $i = 1, \ldots, n$) such that the expected execution time of its jobs is minimized.

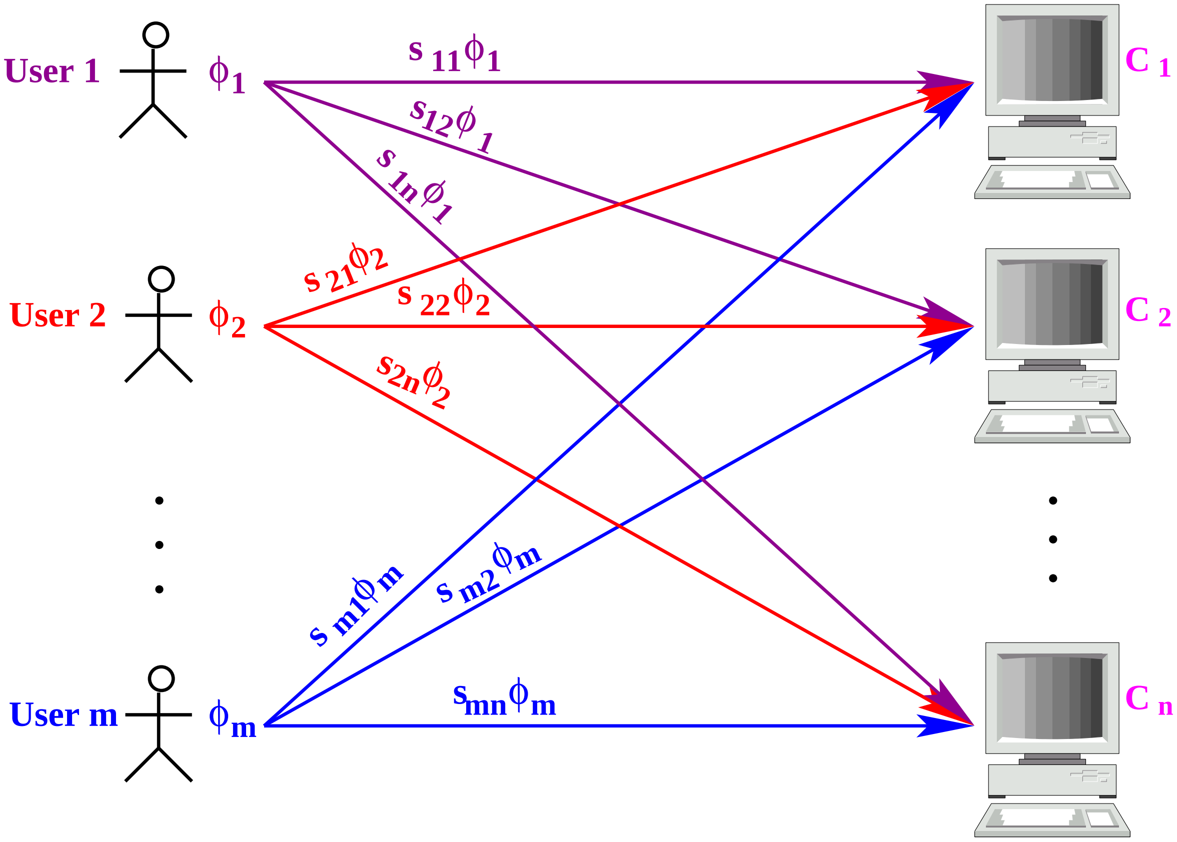

Figure 4.1. The distributed system model.

We formulate this problem as a noncooperative game among users under the assumption that users are 'selfish'. This means that they minimize the expected response time of their own jobs.

Let s_{ji} be the fraction of jobs that user j sends to computer i. The vector $\mathbf{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$ is called the *load balancing strategy* of user j. The vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ is called the *strategy profile* of the load balancing game.

We assume that each computer is modeled as an M/M/1 queueing system and the expected response time at computer i is given by:

$$F_i(\mathbf{s}) = \frac{1}{\mu_i - \sum_{j=1}^{m} s_{ji}\phi_j} \tag{4.1}$$

Thus the overall expected response time of user j is given by:

$$D_j(\mathbf{s}) = \sum_{i=1}^{n} s_{ji} F_i(\mathbf{s}) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i - \sum_{k=1}^{m} s_{ki}\phi_k} \tag{4.2}$$

The goal of user j is to find a feasible load balancing strategy $\mathbf{s}_j$ such that $D_j(\mathbf{s})$ is minimized.

A feasible load balancing strategy profile $\mathbf{s}$ must satisfy the following restrictions:

(i) Positivity: $s_{ji} \geq 0, i = 1, \ldots, n, j = 1, \ldots, m$;

(ii) Conservation: $\sum_{i=1}^{n} s_{ji} = 1, j = 1, \ldots, m$;

(iii) Stability: $\sum_{j=1}^{m} s_{ji}\phi_j < \mu_i, i = 1, \ldots, n$;

The decision of user j depends on the load balancing decisions of other users since $D_j(\mathbf{s})$ is a function of $\mathbf{s}$. We adopt here the Nash equilibrium as the optimality concept [50].

Definition 4.1 (Nash equilibrium) A *Nash equilibrium* of the load balancing game defined above is a strategy profile $\mathbf{s}$ such that for every user j ($j = 1, \ldots, m$):

$$\mathbf{s}_j \in \arg\min_{\tilde{\mathbf{s}}_j} D_j(\mathbf{s}_1, \ldots, \tilde{\mathbf{s}}_j, \ldots, \mathbf{s}_m) \tag{4.3}$$

In other words a strategy profile $\mathbf{s}$ is a Nash equilibrium if no user can benefit by deviating unilaterally from its load balancing strategy to another feasible one.

Remark: In general Nash equilibria are defined in terms of mixed strategies which are probability distributions over the set of pure strategies. In this dissertation we are interested only on pure strategy equilibria.

Similar games were studied in the context of flow control and routing in networks. Orda *et al.* [110] proved that if the expected response time functions are continuous, convex and increasing there exists a unique Nash equilibrium for the game. The closest work to our study is that of Korilis *et al.* [76] in which a similar game is studied in the context of capacity allocation in networks. They studied the structure and properties of the Nash equilibrium for the game. Here, we are interested in finding a way to compute the Nash equilibrium for our load balancing noncooperative game.

We need to determine the strategy profile of user j which must be optimal with respect to the other users strategies. Let $\mu_i^j = \mu_i - \sum_{k=1,k\neq j}^{m} s_{ki}\phi_k$ be the *available processing* rate at processor i as seen by user j. The problem of computing the optimal strategy of user j ($j = 1, \dots, m$) reduces to computing the optimal strategy for a system with n processors having μ_i^j ($i = 1, \dots, n$) as processing rates and ϕ_j as the job arrival rate in the system.

For user j, the associated optimization problem ($\mathrm{OP_j}$) can be described as follows:

$$\min_{\mathbf{s}_j} D_j(\mathbf{s}) \tag{4.4}$$

subject to the constraints:

$$s_{ji} \geq 0, \qquad i = 1, \dots, n \tag{4.5}$$

$$\sum_{i=1}^{n} s_{ji} = 1 \tag{4.6}$$

$$\sum_{k=1}^{m} s_{ki}\phi_k < \mu_i, \qquad i = 1, \dots, n \tag{4.7}$$

Remark: The strategies of all the other users are kept fixed, thus the variables involved in $\mathrm{OP_j}$ are the load fractions of user j, i.e. $\mathbf{s}_j = (s_{j1}, s_{j2}, \dots, s_{jn})$.

There exist some algorithms for finding the solution for similar optimization problems using different objective functions based on Lagrange multipliers. One was proposed by Tantawi and Towsley [128] but it is complex and involves a numerical method for solving a nonlinear equation. Ni and Hwang [104] studied a similar problem for a system with preassigned arrival rates at each computer, but they considered a different objective function from ours. Another one which inspired our approach was proposed by Tang and Chanson [127]. They considered a system were there is only one user that has to assign jobs to computers such that the job execution time for all jobs is minimized. We extend the work in [127] considering a different model in which the influence of the other users' decisions on the optimum is taken into account. Thus we use the same objective function but we use different constraints. Here, we propose an algorithm for computing the optimum considering our model.

We can characterize the optimal solution of $\mathrm{OP_j}$ as follows:

Theorem 4.1 **(OP_j solution)** Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geq \mu_2^j \geq \ldots \geq \mu_n^j$), the solution $\mathbf{s}_j$ of the optimization problem $\mathrm{OP_j}$ is given by:

$$s_{ji} = \begin{cases} \frac{1}{\phi_j}\left(\mu_i^j - \sqrt{\mu_i^j}\frac{\sum_{i=1}^{c_j}\mu_i^j - \phi_j}{\sum_{i=1}^{c_j}\sqrt{\mu_i^j}}\right) & if \quad 1 \leq i < c_j \\ 0 & if \quad c_j \leq i \leq n \end{cases} \tag{4.8}$$

where c_j is the minimum index that satisfies the inequality:

$$\sqrt{\mu_{c_j}^j} \leq \frac{\sum_{k=1}^{c_j}\mu_k^j - \phi_j}{\sum_{k=1}^{c_j}\sqrt{\mu_k^j}} \tag{4.9}$$

Proof: In Appendix B.

Based on the above theorem we derived the following algorithm for solving user j's optimization problem $\mathrm{OP_j}$.

```
BEST-REPLY($\mu_1^j, \ldots, \mu_n^j$, $\phi_j$)
Input: Available processing rates: $\mu_1^j$, $\mu_2^j$, ... $\mu_n^j$;
        Total arrival rate: $\phi_j$
Output: Load fractions: $s_{j1}$, $s_{j2}$, ... $s_{jn}$;

        1. Sort the computers in decreasing order of their available
                processing rates ($\mu_1^j \geq \mu_2^j \geq \ldots \geq \mu_n^j$);
```

```
2. $t \leftarrow \frac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$
3. while ( $t \geq \sqrt{\mu_n^j}$ ) do
       $s_{jn} \leftarrow 0$
       $n \leftarrow n - 1$
       $t \leftarrow \frac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$
4. for $i = 1, \ldots, n$ do
       $s_{ji} \leftarrow \left( \mu_i^j - t\sqrt{\mu_i^j} \right) \frac{1}{\phi_j}$
```

The following theorem proves the correctness of this algorithm.

Theorem 4.2 (Correctness) The load balancing strategy $\{s_{j1}, s_{j2}, \ldots, s_{jn}\}$ computed by the BEST-REPLY algorithm solves the optimization problem $\mathrm{OP_j}$ and is the optimal strategy for user j.

Proof: In Appendix B.

Remarks: (1) The execution time of this algorithm is $O(n \log n)$. This is due to the sorting procedure in step 1. (2) To execute this algorithm each user needs to know the available processing rate at each computer and its job arrival rate. The available processing rate can be determined by statistical estimation of the run queue length of each processor.

Example 5.1:

Let's apply the BEST-REPLY algorithm to a system of 3 computers and only one user. The computers have the following available processing rates: $\mu_1^1 = 10.0$ jobs/sec, $\mu_2^1 = 2.0$ jobs/sec and $\mu_3^1 = 1.0$ jobs/sec. The total arrival rate is $\phi_1 = 6.0$ jobs/sec. The computers are already sorted in decreasing order of their processing rate and we execute Step 2 in which we compute t.

$$t = \frac{10 + 2 + 1 - 6}{\sqrt{10} + \sqrt{2} + \sqrt{1}} = 1.255$$

The while loop in Step 3 is executed because $t > \sqrt{\mu_3^1}$. In this loop $s_{13} = 0$, $n = 2$ and t is updated to $1.311 < \sqrt{\mu_2^1}$ and then the algorithm proceeds to Step 4. In this step the values of the load fractions are computed: $s_{11} = 0.975$ and $s_{12} = 0.025$.

□

4.3 A distributed load balancing algorithm

The computation of Nash equilibrium may require some coordination between the users. From the practical point of view we need decentralization and this can be obtained by using distributed greedy best response algorithms [11]. In these algorithms each user updates from time to time its load balancing strategy by computing the best response against the existing load balancing strategies of the other users. In our case each user updates its strategy in a round-robin fashion.

Based on the BEST-REPLY algorithm presented in the previous section, we devised the following greedy best reply algorithm for computing the Nash equilibrium for our noncooperative load balancing game.

We use the following notations in addition to those of Section 4.2:

j - the user number;

l - the iteration number;

$\mathbf{s}_j^{(l)}$ - the strategy of user j computed at iteration l;

$D_j^{(l)}$ - user j's expected execution time at iteration l;

ϵ - a properly chosen acceptance tolerance;

Send(j, (p, q, r)) - send the message (p, q, r) to user j;

Recv(j, (p, q, r)) - receive the message (p, q, r) from user j;

(where p is a real number, and q, r are integer numbers).

NASH distributed load balancing algorithm:

```
User $j$, ($j = 1, \ldots, m$) executes:
    1. Initialization:
            $\mathbf{s}_j^{(0)} \leftarrow \mathbf{0}$;
            $\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;
            $l \leftarrow 0$;
            $norm \leftarrow 1$;
            $sum \leftarrow 0$;
            $tag \leftarrow \mathrm{CONTINUE}$;
```

```
        $left = [(j-2)\mathrm{mod} m] + 1$;
        $right = [j \mathrm{mod} m] + 1$;
    2. **while** ( 1 ) **do**
        **if**($j = 1$) {user 1}
            **if** ($l \neq 0$)
                **Recv**($left$, ($norm$, $l$, $tag$));
                **if** ($norm < \epsilon$)
                    **Send**($right$, ($norm$, $l$, STOP));
                    **exit**;
            $sum \leftarrow 0$;
            $l \leftarrow l + 1$;
        **else** {the other users}
            **Recv**($left$, ($sum$, $l$, $tag$));
            **if** ($tag$ = STOP)
                **if** ($j \neq m$)    **Send**($right$, ($sum$, $l$, STOP));
                **exit**;
        **for** $i = 1, \ldots, n$ **do**
            Obtain $\mu_i^j$ by inspecting the run queue of each computer
            $(\mu_i^j \leftarrow \mu_i - \sum_{k=1,k\neq j}^{m} s_{ki}\phi_k)$;
        $\mathbf{s}_j^{(l)} \leftarrow$ BEST-REPLY($\mu_1^j, \ldots, \mu_n^j$, $\phi_j$);
        Compute $D_j^{(l)}$;
        $sum \leftarrow sum + |D_j^{(l-1)} - D_j^{(l)}|$;
        **Send**($right$, ($sum$, $l$, CONTINUE));
    **endwhile**
```

The execution of this algorithm is restarted periodically or when the system parameters are changed. Once the Nash equilibrium is reached, the users will continue to use the same strategies and the system remains in equilibrium. This equilibrium is maintained until a new execution of the algorithm is initiated. The running time of one iteration of the NASH algorithm is $O(nlogn)$.

An important practical question is whether such 'best reply' algorithms converge to the Nash equilibrium. The only known results about the convergence of such algorithms have been obtained in the context of routing in parallel links. These studies have been limited to special cases of two parallel links shared by two users [110] or by $m \geq 2$ users but with linear cost links [6]. For M/M/1 type cost functions there is no known proof that such algorithms converge for more than two users. As shown by several experiments done on different settings, these algorithms may converge for more than two users. In the next section we present such experiments that confirm this hypothesis (also see [17]). The convergence proof for more than two users is still an open problem.

4.4 Experimental results

4.4.1 Simulation environment

The simulations were carried out using Sim++ [31], a simulation software package written in C++. This package provides an application programming interface which allows the programmer to call several functions related to event scheduling, queueing, preemption and random number generation. The simulation model consists of a collection of computers connected by a communication network. Jobs arriving at the system are distributed to the computers according to the specified load balancing scheme. Jobs which have been dispatched to a particular computer are *run-to-completion* (i.e. no preemption) in FCFS (first-come-first-served) order.

Each computer is modeled as an M/M/1 queueing system [73]. The main performance metrics used in our simulations are the *expected response time* and the *fairness index*. The *fairness index* (defined from the users' perspective),

$$I(\mathbf{D}) = \frac{[\sum_{j=1}^{m} D_j]^2}{m \sum_{j=1}^{m} D_j^2} \tag{4.10}$$

was proposed in [64] to quantify the fairness of load balancing schemes. Here the parameter $\mathbf{D}$ is the vector $\mathbf{D} = (D_1, D_2, \ldots, D_m)$ where D_j is the total expected execution time of user j's jobs. This index is a measure of the 'equality' of users' total expected job execution times. If all the users have the same total expected job execution times then $I = 1$ and the system is 100% fair to all users and it is load balanced. If the differences on D_j increase, I decreases and the load balancing scheme favors only some users.

The simulations were run over several thousands of seconds, sufficient to generate a total of 1 to 2 millions jobs typically. Each run was replicated five times with different random number streams and the results averaged over replications. The standard error is less than 5% at the 95% confidence level.

4.4.2 Performance evaluation

For comparison purposes we consider three existing static load balancing schemes [24, 67, 71]. A brief description of these schemes is given below:

- **Proportional Scheme (PS)** [24]: According to this scheme each user allocates its jobs to computers in proportion to their processing rate. This allocation seems to be a natural choice but it may not minimize the user's expected response time or the overall expected response time. The fairness index for this scheme is always 1 as can be easily seen from equation (4.10). The running time of PS is $O(mn)$.

- **Global Optimal Scheme (GOS)** [71]: This scheme minimizes the expected execution time over all jobs executed by the system. The load fractions (**s**) are obtained by solving the following nonlinear optimization problem:

  $$\min_{\mathbf{s}} \frac{1}{\Phi} \sum_{k=1}^{m} \phi_j D_j(\mathbf{s}) \tag{4.11}$$

  subject to the constraints:

  $$s_{ji} \geq 0, \qquad i = 1, \ldots, n, \qquad j = 1, \ldots, m \tag{4.12}$$

  $$\sum_{i=1}^{n} s_{ji} = 1, \qquad j = 1, \ldots, m \tag{4.13}$$

  $$\sum_{j=1}^{m} s_{ji}\phi_j < \mu_i, \qquad i = 1, \ldots, n \tag{4.14}$$

  This scheme provides the overall optimum for the expected execution time but it is not user-optimal and is unfair. This is an iterative algorithm and the running time of each iteration is $O(mnlogn + mnlog(1/\epsilon))$, where ϵ denotes the acceptable tolerance.

- **Individual Optimal Scheme (IOS)** [67]: In this scheme each job optimizes its response time for itself independently of others. In general the Wardrop equilibrium, which is the solution given by this scheme, is not optimal and in some cases we expect worse response time than the other policies [67]. It is based on an iterative procedure that is not very efficient. For a complete description of IOS algorithm see [67]. The advantage of this scheme is that it provides a fair allocation. This is an

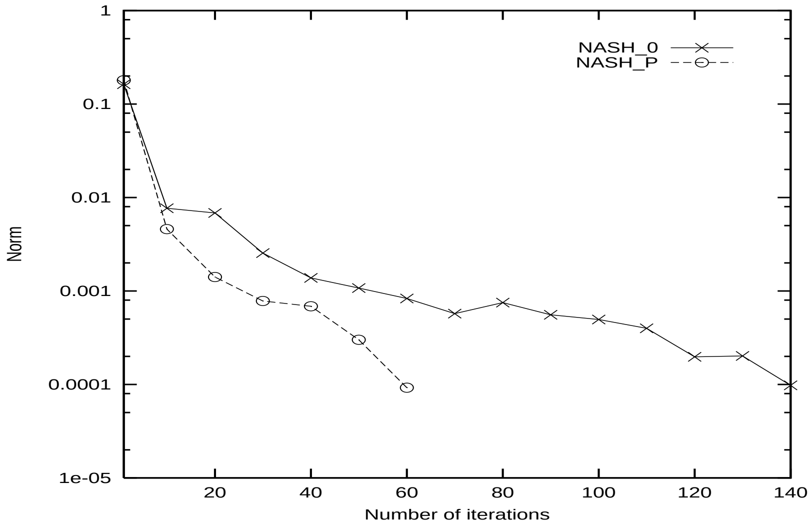

Figure 4.2. Norm vs. number of iterations.

iterative algorithm and the running time of each iteration is $O(mnlogn + mnlog(1/\epsilon))$, where ϵ denotes the acceptable tolerance.

Remark: Among the three schemes described above, the IOS scheme is the only scheme that is based on game theoretic concepts.

We evaluated the schemes presented above under various system loads and configurations. Also the convergence of the NASH load balancing algorithm is investigated. In the following we present and discuss the simulation results.

The convergence of NASH algorithm

An important issue related to the greedy best reply algorithm presented above is the dynamics of reaching the equilibrium. We consider first the NASH algorithm using $\mathbf{s}^{(0)} = \mathbf{0}$ as the initialization step. This variant of the algorithm will be called NASH_0. This initialization step is an obvious choice but it may not lead to a fast convergence to the equilibrium.

We propose a variant of the algorithm in which the initialization step is replaced by:

```
1. Initialization:
      for $i = 1, \dots, n$ do
            $s_{ji}^{(0)} \leftarrow \frac{\mu_i}{\sum_{k=1}^{n} \mu_k}$;
```

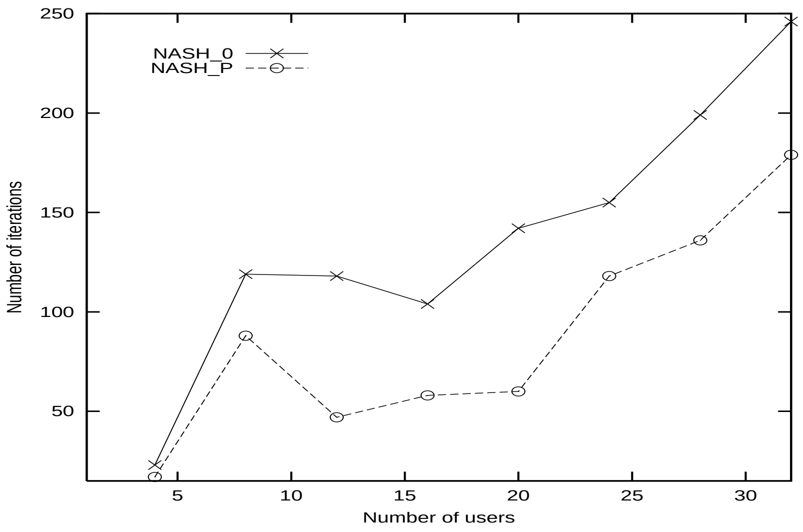

Figure 4.3. Convergence of best reply algorithms (until $norm < 10^{-4}$).

$\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;
$l \leftarrow 0$;
$\vdots$

We call this new version NASH_P. Using this initialization step the starting point will be a proportional allocation of jobs to computers according to their processing rate. We expect a better convergence using NASH_P instead of NASH_0. To study the convergence of these algorithms we consider a system with 16 computers shared by 10 users. The norm vs. the number of iterations is shown in Figure 4.2. It can be seen that the NASH_P algorithm significantly outperforms NASH_0 algorithm. The intuitive explanation for this performance is that the initial proportional allocation is close to the equilibrium point and the number of iterations needed to reach the equilibrium is reduced. Using the NASH_P algorithm the number of iterations needed to reach the equilibrium is reduced by more than a half compared with NASH_0.

Next, we study the influence of the number of users on the convergence of both algorithms. In Figure 4.3 we present the number of iterations needed to reach the equilibrium ($norm < 10^{-4}$) for a system with 16 computers and a variable number of users (from 4 to 32). It can be observed that NASH_P significantly outperforms NASH_0 reducing the number of iterations needed to reach the equilibrium in all the cases.

Remark: We now present the execution times of each load balancing scheme. The execution time of the NASH algorithm in the case of medium system load, 16 computers and 10 users is about 12.5 msec per

Table 4.1. System configuration.

Relative processing rate	1	2	5	10
Number of computers	6	5	3	2
Processing rate (jobs/sec)	10	20	50	100

iteration, on a SUN Ultra 10 workstation (440Mhz). Considering the same parameters, one iteration of GOS algorithm takes 42 sec and one iteration of IOS takes 47 sec. These high execution times for GOS and IOS are due to the high complexity of operations involved in these two algorithms (in computing the solutions of nonlinear equations). For the NASH algorithm each user computations have lower complexity than the other schemes and are performed concurrently. The number of iterations for medium loads and 10 users are, GOS: 3; IOS: 3; NASH: 40.

Effect of system utilization

To study the effect of system utilization we simulated a heterogeneous system consisting of 16 computers with four different processing rates. This system is shared by 10 users. In Table 4.1, we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We consider only computers that are at most ten times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

In Figure 4.4, we present the expected response time of the system and the fairness index for different values of system utilization (ranging from 10% to 90%). *System utilization* (ρ) is defined as the ratio of the total arrival rate to the aggregate processing rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i} \tag{4.15}$$

It can be observed that at low loads (ρ from 10% to 40%) all the schemes except PS yield almost the

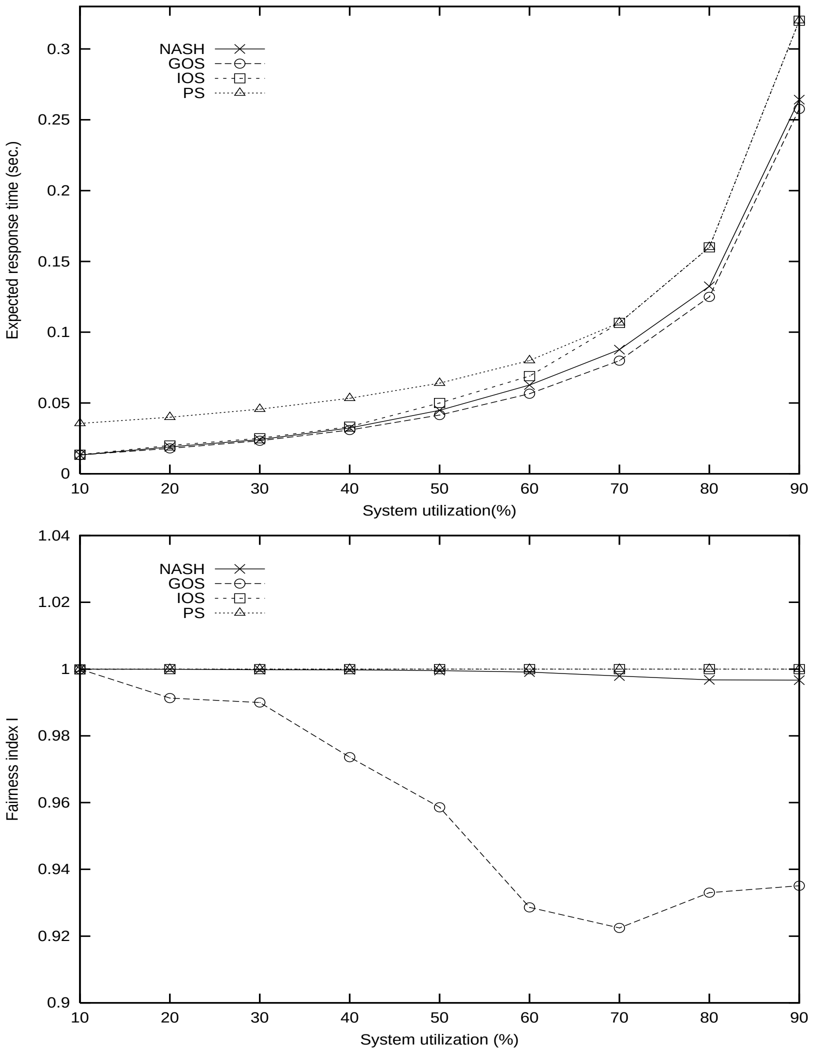

Figure 4.4. The expected response time and fairness index vs. system utilization.

same performance. The poor performance of PS scheme is due to the fact that the less powerful computers are significantly overloaded.

At medium loads (ρ from 40% to 60%) NASH scheme performs significantly better than PS and approaches the performance of GOS. For example at load level of 50% the expected response time of NASH is 30% less than PS and 7% greater than GOS.

At high loads IOS and PS yield the same expected response time which is greater than that of GOS and NASH. The expected response time of NASH scheme is very close to that of GOS.

The PS and IOS schemes maintain a fairness index of 1 over the whole range of system loads. It can

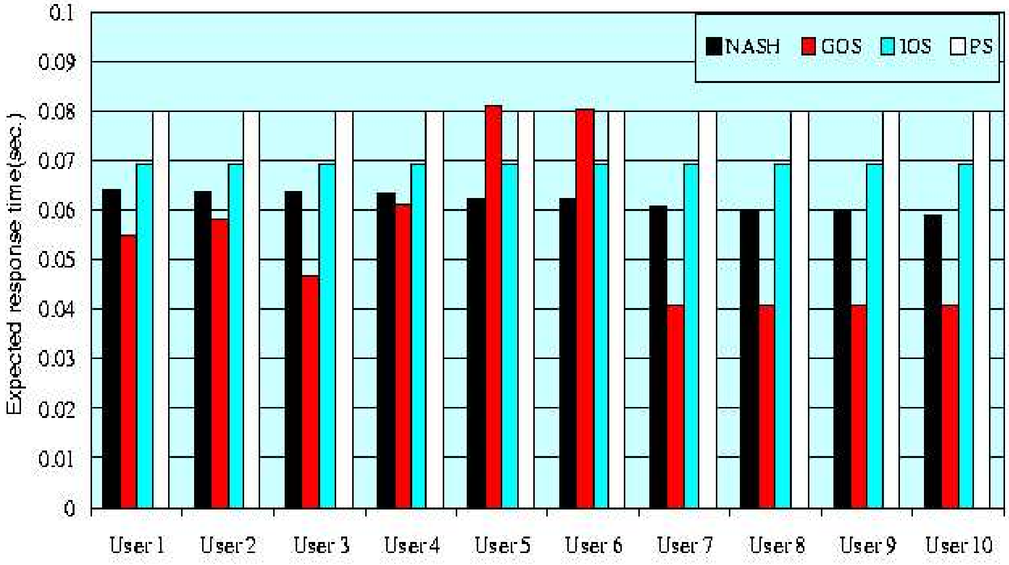

Figure 4.5. Expected response time for each user.

be shown that the PS has a fairness index of 1 which is a constant independent of the system load. The fairness index of GOS varies from 1 at low load, to 0.92 at high load. The NASH scheme has a fairness index close to 1 and each user obtains the minimum possible expected response time for its own jobs (i.e. it is user-optimal). User-optimality and decentralization are the main advantages of NASH scheme.

An interesting issue is the impact of static load balancing schemes on individual users. In Figure 4.5, we present the expected response time for each user considering all static schemes at medium load (ρ=60%). The PS and IOS schemes guarantee equal expected response times for all users but with the disadvantage of a higher expected execution time for their jobs. It can be observed that in the case of GOS scheme there are large differences in users' expected execution times. NASH scheme provides the minimum possible expected execution time for each user given what every other users are doing (according to the properties of the Nash equilibrium).

Effect of heterogeneity

In a distributed system, heterogeneity usually consists of: processor speed, memory and I/O. A simple way to characterize system heterogeneity is to use the processor speed. Furthermore, it is reasonable to

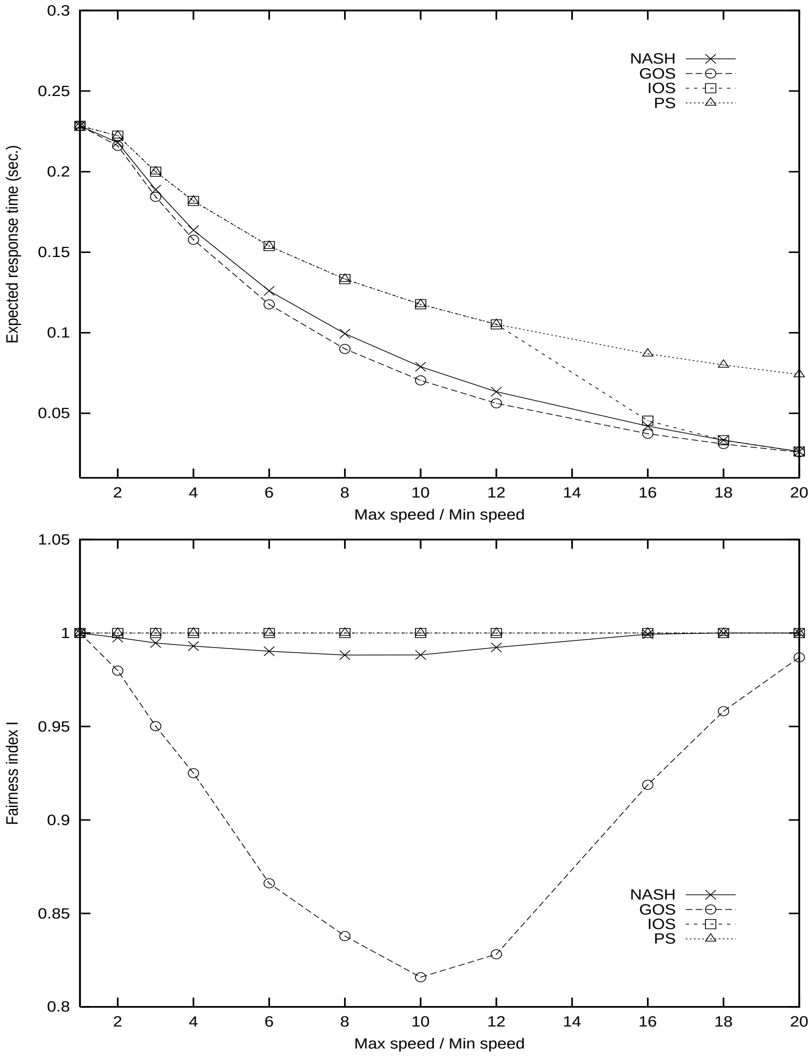

Figure 4.6. The effect of heterogeneity on the expected response time and fairness index.

assume that a computer with high speed processor will have matching resources (memory and I/O). One of the common measures of heterogeneity is the *speed skewness* which is defined as the ratio of maximum processing rate to minimum processing rate of the computers. This measure is somehow limited but for our goals it is satisfactory.

In this section, we investigate the effectiveness of load balancing schemes by varying the speed skewness. We simulate a system of 16 heterogeneous computers: 2 fast and 14 slow. The slow computers have a relative processing rate of 1 and we varied the relative processing rate of the fast computers from 1 (which correspond to a homogeneous system) to 20 (which correspond to a highly heterogeneous system). The

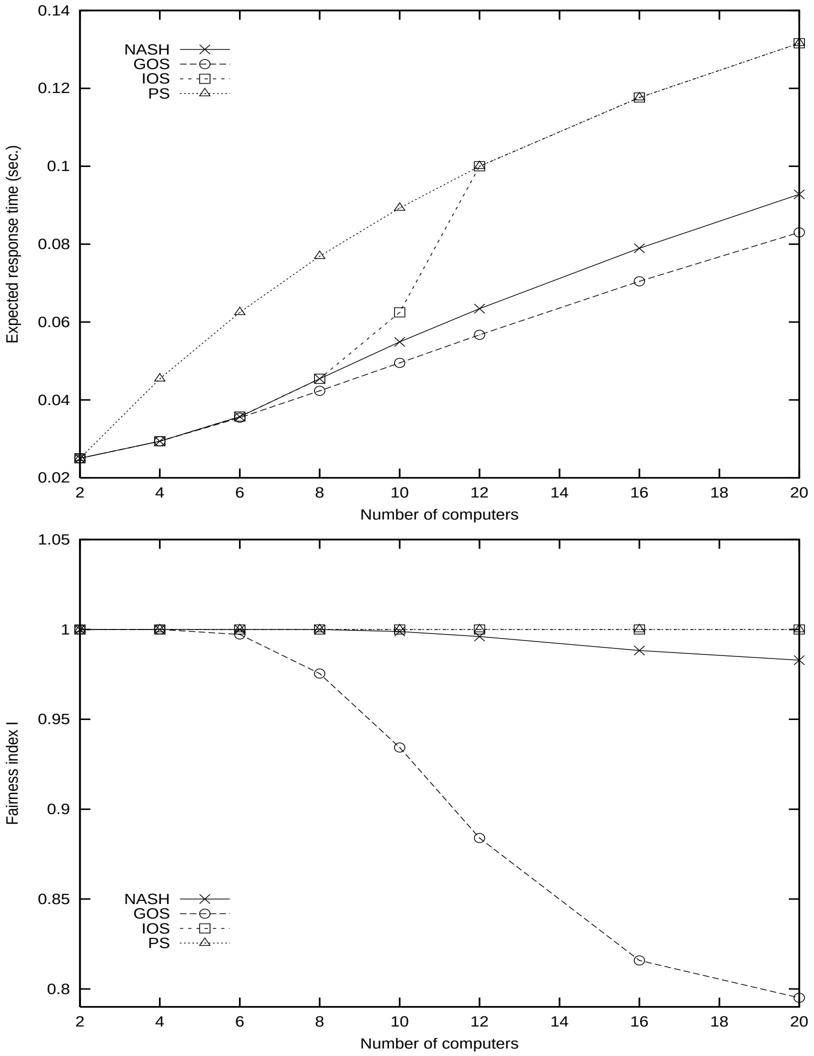

Figure 4.7. The effect of system size on the expected response time and fairness index.

system utilization was kept constant $\rho = 60\%$.

In Figure 4.6, we present the effect of speed skewness on the expected response time and fairness. It can be observed that increasing the speed skewness the GOS and NASH schemes yield almost the same expected response time which means that in highly heterogeneous systems the NASH scheme is very effective. NASH scheme has the additional advantage of decentralization and user-optimality which is very important in actual distributed systems. PS scheme performs poorly because it overloads the slowest computers. The IOS scheme performs well at high speed skewness approaching the performance of NASH and GOS, but at low speed skewness it performs poorly.

Effect of system size

An important issue is to study the influence of system size on the performance of load balancing schemes. To study this issue we simulated a heterogeneous distributed system consisting of two types of computers: slow computers (relative processing rate = 1) and fast computers (relative processing rate = 10). Figure 4.7 shows the expected response time and the fairness index where the number of computers increases from 2 (fast computers only) to 20 (2 fast and 18 slow computers). The performance of NASH and GOS is almost the same when we have few computers (2 to 8). The PS scheme performs poorly even for a small system. The expected response time for IOS degrades increasing the system size and approaches the expected response time of PS. NASH guarantees a good performance for medium and large systems and the same performance as GOS for small systems. The additional advantage of NASH is that it is a distributed schemes providing a user-optimal allocation.

Experiments using the hyper-exponential distribution for the inter-arrival times

We investigate the performance of our load balancing scheme by considering a job arrival process different from the Poisson distribution. We use the two-stage hyper-exponential distribution for modeling the job arrival process [73]. The parameters of the system used in the simulation are those presented in Table 4.1. The coefficient of variation for the job inter-arrival time is set to 1.6.

In Figure 4.8, we present the expected response time of the system for different values of system utilization (ranging from 10% to 90%). It can be observed that the performance is similar to that obtained using the Poisson distribution for the arrival process. At low loads (ρ from 10% to 40%) all the schemes except PS yield almost the same performance. The poor performance of PS scheme is due to the fact that the less powerful computers are significantly overloaded. At medium loads (ρ from 40% to 60%) the NASH scheme performs significantly better than PS and approaches the performance of GOS. For example at load level of 50% the expected response time of NASH is 23% lower than PS and 15% higher than GOS. At high loads IOS and PS yield a higher expected response time than that of GOS and NASH. At the load level of 90% the expected response time of the NASH scheme is 18% higher than GOS and 19% lower than PS.

The PS and IOS schemes maintain a fairness very close to 1 over the whole range of system loads. The

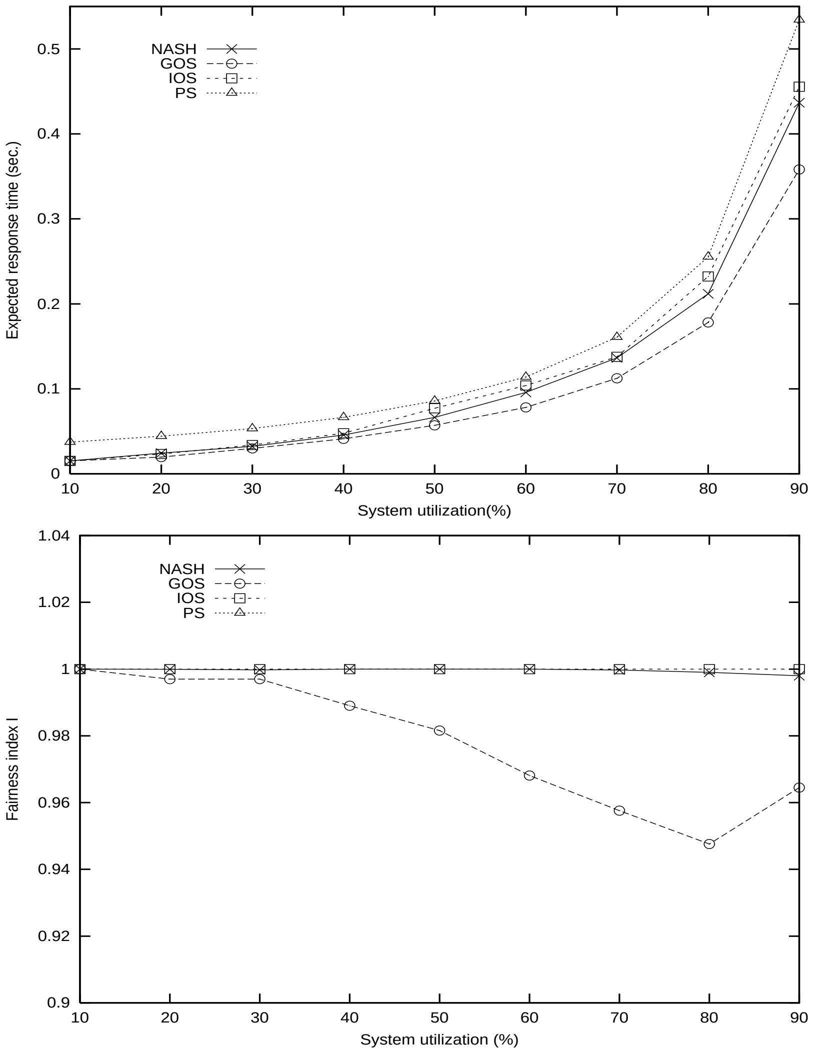

Figure 4.8. Expected response time and fairness (hyper-exponential distribution of arrivals).

fairness index of GOS varies from 1 at low loads, to 0.94 at high loads. The NASH scheme has a fairness index close to 1 and each user obtains the minimum possible expected response time for her own jobs (i.e. it is user-optimal).

4.5 Conclusion

In this chapter we have presented a game theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. We formulated the load balancing problem in heterogeneous distributed systems as a noncooperative game among users. For this game the Nash equilibrium provides an user-optimal operation point for the distributed system. For the proposed noncooperative load balancing game, we presented the structure of the Nash equilibrium. Based on this structure we derived a new distributed algorithm for computing it. We compared the performance of our noncooperative load balancing scheme with other existing schemes. The main advantages of our noncooperative load balancing scheme are its distributed structure and user-optimality.

Chapter 5

Algorithmic Mechanism Design for Load Balancing

5.1 Introduction

In current distributed systems such as computational grids, resources belong to different self interested agents or organizations. These agents may manipulate the load allocation algorithm in their own benefit and their selfish behavior may lead to severe performance degradation and poor efficiency. Solving such problems involving selfish agents is the object of *mechanism design theory* (also called *implementation theory*) [111]. This theory helps design mechanisms (protocols) in which the agents are always forced to tell the truth and follow the rules. Such mechanisms are called *truthful* or *strategy-proof*.

Each participating agent has a privately known function called *valuation* which quantifies the agent benefit or loss. The valuation depends on the outcome and is reported to a centralized mechanism. The mechanism chooses an outcome that maximizes a given objective function and makes payments to the agents. The valuations and payments are expressed in some common unit of currency. The payments are designed and used to motivate the agents to report their true valuations. Reporting the true valuations leads to an optimal value for the objective function. The goal of each agent is to maximize the sum of her valuation and payment.

In this chapter we consider the mechanism design problem for load balancing in distributed systems. Our goal is to design a mechanism that uses the optimal load balancing algorithm. The optimal algorithm belongs to the global approach in the classification presented in Chapter 2. To design our mechanism we use the framework derived by Archer and Tardos in [7]. We assume that each computer in the distributed system is characterized by its processing rate and only computer i knows the true value of its processing rate. Jobs arrive at the system with a given arrival rate. The optimal algorithm finds the fraction of load that is allocated to each computer such that the expected execution time is minimized. The *cost* incurred by each computer is proportional to its utilization. The mechanism will ask each agent (computer) to report its processing rate and then compute the allocation using the optimal algorithm. After computing the allocation the mechanism hands payments to computers.

Each computer goal is to chose a processing rate to report to the mechanism such that its profit is maximized. The *profit* is the difference between the payment handed by the mechanism and the true cost of processing the allocated jobs. The payments handed by the mechanism must motivate the computers to report their true value such that the expected response time of the entire system is minimized. Thus we need to design a payment function that guarantees this property.

Related work

Recently, with the emergence of the Internet as a global platform for computation and communication, the need for efficient protocols that deals with self-interested agents has increased. This motivated the use of mechanism design theory in different settings such as market based protocols for resource allocation in computational grids [18, 134], market based protocols for scheduling [131], congestion control [70], routing [44] and mechanisms for trading CPU time [105]. A recent survey on distributed algorithmic mechanism design is [46]. For an introduction to general mechanism design theory see [111].

The most important result in mechanism design theory is the Vickrey-Clarke-Groves (VCG) mechanism [25, 55, 129]. The VCG mechanism allows arbitrary form for valuations and its applicability is restricted to utilitarian objective functions (i.e. the objective function is the sum of agents' valuations).

Nisan and Ronen [106] studied the mechanism design problem for several standard problems in computer science. Using the VCG mechanism they solved the shortest path problem in a graph where each edge belongs to a different agent. For scheduling on unrelated machines they designed an n-approximation truthful mechanism, where n is the number of agents. They proved a lower bound of 2 to the approximation ratio of any mechanism for this problem and conjectured that no truthful mechanism can yield an approximation ratio better than n. They also gave a mechanism that solves exactly the problem of scheduling jobs on unrelated machines in a model where the mechanism has more information. In this extended model the payments are given after jobs' execution allowing the mechanism to verify the agent's declarations and penalize them for lying.

The computational feasibility of VCG mechanisms is addressed in [107]. The authors proved that all reasonable approximations or heuristics for a wide class of minimization problems yield non-truthful VCG-based mechanisms (i.e. mechanisms that use the VCG payment scheme and a suboptimal allocation algorithm). They showed that under reasonable assumption any VCG-based mechanism can be made feasible truthful.

A polynomial VCG mechanism for shortest paths is proposed in [59]. Feigenbaum *et al*. [45] studied two mechanisms for cost-sharing in multicast transmissions. Frugality of shortest paths mechanisms is investigated in [8].

Archer and Tardos [7] applied mechanism design theory to several combinatorial optimization problems where agent's secret data is represented by a single real valued parameter. They provided a method to design mechanisms for general objective functions and restricted form for valuations. For scheduling related parallel machines they gave a 3-approximation algorithm and used it to design a truthful mechanism. They also gave truthful mechanisms for maximum flow, scheduling related machines to minimize the sum of completion times, optimizing an affine function and special cases of uncapacitated facility location.

Our contributions

We design a truthful mechanism for solving the static load balancing problem in distributed systems. We prove that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. We derive a protocol that implements our mechanism and present experiments to show its effectiveness.

Organization

This chapter is structured as follows. In Section 5.2 we present the mechanism design terminology. In Section 5.3 we present our distributed system model. In Section 5.4 we design a truthful mechanism for load balancing in distributed systems. In Section 5.5 the effectiveness of our load balancing mechanism is investigated. In Section 5.6 we draw conclusions and present future directions.

5.2 Mechanism Design Concepts

In this section we introduce some important mechanism design concepts. We limit our description to mechanism design problems for one parameter agents. In this type of mechanism design problems each agent has some private data represented by a single real valued parameter [7]. In the following we define such problem.

Definition 5.1 (Mechanism design problem) A *mechanism design problem for one parameter agents* is characterized by:

(i) A finite set Λ of allowed outputs. The output is a vector $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$, $\lambda(b) \in \Lambda$, computed according to the agents' bids, $b = (b_1, b_2, \ldots, b_n)$. Here, b_i is the value (bid) reported by agent i to the mechanism.

(ii) Each agent i, ($i = 1, \ldots, n$), has a privately known parameter t_i called her *true value*. The cost incurred by each agent depends on the output and on her true value and is denoted as $cost_i(t_i, \lambda(b))$.

(iii) Each agent goal is to maximize her profit. The profit of agent i is $profit_i(t_i, b) = P_i(b) - cost_i(t_i, \lambda(b))$, where P_i is the payment handed by the mechanism to agent i.

(iv) The goal of the mechanism is to select an output λ that optimizes a given cost function $g(t, \lambda)$.

We assume that the cost functions have the following particular form: $cost_i(t_i, \lambda(b)) = t_i\lambda_i(b)$, i.e. t_i represents the cost per unit load.

Definition 5.2 (Mechanism) A *mechanism* is characterized by two functions:

(i) The *output function* $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$. This function has as input the vector of agents' bids $b = (b_1, b_2, \ldots, b_n)$ and returns an output $\lambda \in \Lambda$.

(ii) The *payment function* $P(b) = (P_1(b), P_2(b), \ldots, P_n(b))$ that gives the payment handed by the mechanism to each agent.

Notation: In the rest of the chapter we denote by b_{-i} the vector of bids not including the bid of agent i. The vector b is represented as (b_{-i}, b_i).

Definition 5.3 (Truthful mechanism) A mechanism is called *truthful* if for every agent i of type t_i and for every bids b_{-i} of the other agents, the agent's profit is maximized when she declares her real type t_i. (i.e. truth-telling is a dominant strategy).

Definition 5.4 (Truthful payment scheme) We say that an output function admits a *truthful payment scheme* if there exists a payment function P such that the mechanism is truthful.

A desirable property of a mechanism is that the profit of a truthful agent is always non-negative. The agents hope for a profit by participating in the mechanism.

Definition 5.5 (Voluntary participation mechanism) We say that a mechanism satisfies the *voluntary participation condition* if $profit_i(t_i, (b_{-i}, t_i)) \geq 0$ for every agent i, true values t_i, and other agents' bids b_{-i} (i.e. truthful agents never incur a loss).

5.3 Distributed System Model

We consider a distributed system that consists of n heterogeneous computers connected by a communication network. Computers have the same processing capabilities in the sense that a job may be processed from start to finish at any computer in the system. We assume that each computer is modeled as an M/M/1 queueing system (i.e. Poisson arrivals and exponentially distributed processing times) [73] and is characterized by its *average processing rate* μ_i, $i = 1, \ldots, n$. Jobs are generated by users and arrive at the system according to a time invariant Poisson process with average rate Φ. We call Φ the *total job arrival rate* in the system. The total job arrival rate must be less than the aggregate processing rate of the system (i.e. $\Phi < \sum_{i=1}^{n} \mu_i$). The system model is presented in Figure 5.1. The system has to decide on how to distribute jobs to computers such that it will operate optimally. We assume that the decision to distribute jobs to computers is *static* i.e. it does not depend on the current state of the system. Thus we need to find the *load* λ_i that is assigned to computer i ($i = 1, \ldots, n$) such that the expected response time of all jobs is minimized. The expected response time at computer i is given by:

$$F_i(\lambda_i) = \frac{1}{\mu_i - \lambda_i} \tag{5.1}$$

Thus the overall expected response time is given by:

$$D(\lambda) = \frac{1}{\Phi}\sum_{i=1}^{n} \lambda_i F_i(\lambda_i) = \frac{1}{\Phi}\sum_{i=1}^{n} \frac{\lambda_i}{\mu_i - \lambda_i} \tag{5.2}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the vector of loads assigned to computers.

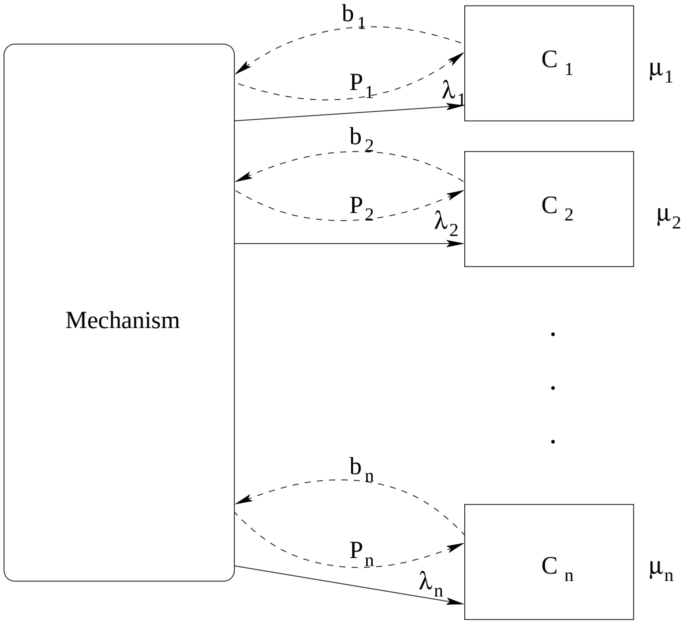

Figure 5.1. The distributed system model.

We assume that computers are agents and each of them has a *true value* t_i represented by the inverse of its processing rate, $t_i = \frac{1}{\mu_i}$. Only computer i knows t_i. The mechanism will ask each computer i to report its value b_i (the inverse of its processing rate). The computers may not report the true value. After all the computers report their values the mechanism computes an output function (i.e. the loads assigned to computers), $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$, according to the agents' bids such that the overall expected execution time is minimized. The mechanism also hands a payment $P_i(b)$ to each computer. All computers know the algorithm used to compute the output function (allocation) and the payment scheme.

Each computer incurs some cost:

$$cost_i(t_i, \lambda(b)) = t_i \lambda_i(b) \tag{5.3}$$

The cost is equivalent to computer utilization. The greater the utilization, the greater the cost. We assume each computer wants to choose its strategy (what value b_i to report) such that its profit is maximized. The profit for each computer is defined as the payment received from the mechanism minus the cost incurred in running the jobs allocated to it:

$$profit_i(t_i, b) = P_i(b) - cost_i(t_i, \lambda(b)) \tag{5.4}$$

Our goal is to design a truthful mechanism that minimizes the overall expected response time of the system. This involves finding an allocation algorithm and a payment scheme that minimizes the overall expected response time according to the computer bids b_i and motivates all the computers to report their true values t_i.

5.4 Designing the Mechanism

To design our load balancing mechanism we use the framework proposed by Archer and Tardos in [7]. They provided a method to design mechanisms where each agent's true data is represented by a single real valued parameter. According to this method, to obtain a truthful mechanism we must find an output function satisfying two conditions: (a) it minimizes $D(\lambda)$ and, (b) it is decreasing in the bids. In addition, we want a mechanism satisfying voluntary participation. To guarantee this property we must find a payment function satisfying voluntary participation.

First we are interested in finding an output function $\lambda(b)$ that minimizes the expected execution time over all jobs, $D(\lambda)$, and produces a feasible allocation. Then we will show that this output function is decreasing in the bids.

Definition 5.6 (Feasible allocation) A *feasible allocation* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a load allocation that satisfies the following conditions:

(i) Positivity: $\lambda_i \geq 0, i = 1, \ldots, n$;

(ii) Conservation: $\sum_{i=1}^{n} \lambda_i = \Phi$;

(iii) Stability: $\lambda_i < \mu_i$, $i = 1, \ldots, n$.

The optimal load allocation can be obtained solving the following nonlinear optimization problem:

$$\min_{\lambda} D(\lambda) \tag{5.5}$$

subject to the constraints defined by the conditions i)-iii).

Thus, obtaining the solution to this problem involves minimizing the convex function $D(\lambda)$ over a convex feasible region defined by the conditions i)-iii). In this case the first order Kuhn-Tucker conditions are necessary and sufficient for optimality [94].

Let $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [94]. The Lagrangian function is:

$$L(\lambda_1, \lambda_2, \ldots, \lambda_n, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \lambda_i F_i(\lambda_1) -$$

$$\alpha(\sum_{i=1}^{n} \lambda_i - \Phi) - \sum_{i=1}^{n} \eta_i \lambda_i \tag{5.6}$$

The Kuhn-Tucker conditions imply that λ_i, $i = 1, \ldots, n$ is the optimal solution to our problem if and only if there exists $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ such that:

$$\frac{\partial L}{\partial \lambda_i} = 0 \tag{5.7}$$

$$\frac{\partial L}{\partial \alpha} = 0 \tag{5.8}$$

$$\eta_i \lambda_i = 0, \quad \eta_i \geq 0, \quad \lambda_i \geq 0, \qquad i = 1, \ldots, n \tag{5.9}$$

These conditions become:

$$\lambda_i F_i^{'}(\lambda_i) + F_i(\lambda_i) - \alpha - \eta_i = 0, \qquad i = 1, \ldots, n \tag{5.10}$$

$$\sum_{i=1}^{n} \lambda_i = \Phi \tag{5.11}$$

$$\eta_i \lambda_i = 0, \quad \eta_i \geq 0, \quad \lambda_i \geq 0, \qquad i = 1, \ldots, n \tag{5.12}$$

These are equivalent to:

$$\alpha = \lambda_i F_i^{'}(\lambda_i) + F_i(\lambda_i), \quad \text{if} \quad \lambda_i > 0 \qquad 1 \leq i \leq n \tag{5.13}$$

$$\alpha \leq \lambda_i F_i^{'}(\lambda_i) + F_i(\lambda_i), \quad \text{if} \quad \lambda_i = 0 \qquad 1 \leq i \leq n \tag{5.14}$$

$$\sum_{i=1}^{n} \lambda_i = \Phi, \quad \lambda_i \geq 0, \quad i = 1, \ldots, n \tag{5.15}$$

By solving these conditions one can obtain the optimal algorithm but this is not our focus in this chapter. Algorithms for this kind of optimization problem were proposed in the past [127, 128]. For the clarity of presentation we describe a variant of the algorithm in [127] using our notations. Our derivation of this algorithm is different from their approach and was partially presented above because some of the equations will be used to prove our results in Theorem 5.1 and Theorem 5.2 below. We now present the algorithm.

OPTIM algorithm:

```
Input: Bids submitted by computers: $b_1, b_2, \ldots b_n$;
          Total arrival rate: $\Phi$
Output: Load allocation: $\lambda_1, \lambda_2, \ldots \lambda_n$;
          1. Sort the computers in increasing order of
                 their bids ($b_1 \leq b_2 \leq \ldots \leq b_n$);
          2. $c \leftarrow \frac{\sum_{i=1}^{n} 1/b_i - \Phi}{\sum_{i=1}^{n} \sqrt{1/b_i}}$;
          3. while ( $c > \sqrt{1/b_n}$ ) do
                 $\lambda_n \leftarrow 0$;
                 $n \leftarrow n - 1$;
                 $c \leftarrow \frac{\sum_{i=1}^{n} 1/b_i - \Phi}{\sum_{i=1}^{n} \sqrt{1/b_i}}$;
          4. for $i = 1, \ldots, n$ do
                 $\lambda_i \leftarrow 1/b_i - c\sqrt{1/b_i}$;
```

Remark: In step 2 the coefficient c is computed. This coefficient is used in step 3 (while-loop) to determine which computers have slow processing rates. If $c > \sqrt{1/b_n}$ then computer C_n is too slow and will not be used for the allocation (i.e. $\lambda_n = 0$). The number of computers n is decremented by 1 and a new c is computed. Step 4 computes the loads λ_i that will be allocated to each computer.

This algorithm computes an allocation $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ that provides the overall optimum for the expected execution time. This optimum is obtained according to the bids reported by computers. If some of the computers declare values different than their true values ($t_i = 1/\mu_i$), this optimum may not be the same as the 'true optimum' obtained when all the computers declare their true values. So if some of the computers lie we expect worse performance (i.e. higher overall expected execution time).

We found an output function (given by OPTIM) that minimizes $D(\lambda)$. Now we must prove that this output function admits a truthful mechanism. In the following we state and prove two theorems: (i) that the output function is decreasing in the bids (thus guaranteeing truthfulness) and, (ii) that our mechanism admits a truthful payment scheme satisfying voluntary participation.

Definition 5.7 (Decreasing output function) An output function $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ is *decreasing* if each $\lambda_i(b_{-i}, b_i)$ is a decreasing function of b_i for all i and b_{-i}.

Theorem 5.1 The output function $\lambda(b)$ computed by the optimal algorithm is decreasing.

Proof: In Appendix C.

Theorem 5.2 The output function $\lambda(b)$ computed by the optimal algorithm admits a *truthful* payment scheme satisfying *voluntary participation* and the payment for each computer i ($i = 1, 2, \ldots, n$) is given by:

$$P_i(b_{-i}, b_i) = b_i \lambda_i(b_{-i}, b_i) + \int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx \tag{5.16}$$

□

Proof: In Appendix C.

Remarks: Analogously to [7] we obtained $P_i(b_{-i}, b_i)$ for our mechanism. The first term, $b_i \lambda_i(b_{-i}, b_i)$, of the payment function in equation (5.16) compensates the cost incurred by computer i. The second term, $\int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx$ represents the expected profit of computer i. If computer i bids its true value, t_i, then its profit P_t is:

$$P_t = profit_i(t_i, (b_{-i}, t_i)) = t_i\lambda_i(b_{-i}, t_i) + \int_{t_i}^{\infty} \lambda_i(b_{-i}, x)dx$$

$$-t_i\lambda_i(b_{-i}, t_i) = \int_{t_i}^{\infty} \lambda_i(b_{-i}, x)dx$$

If computer i bids its true value then the expected profit is greater than in the case it bids other values. We can explain this as follows. If computer i bids higher ($b_i^h > t_i$) then the expected profit P_h is:

$$P_h = profit_i(t_i, (b_{-i}, b_i^h)) = b_i^h\lambda_i(b_{-i}, b_i^h)$$

$$+ \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx - t_i\lambda_i(b_{-i}, b_i^h)$$

$$= (b_i^h - t_i)\lambda_i(b_{-i}, b_i^h) + \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx$$

Because $\int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx < \infty$ and $b_i^h > t_i$ we can express the profit when computer i bids the true value as follows:

$$P_t = \int_{t_i}^{b_i^h} \lambda_i(b_{-i}, x)dx + \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx$$

Because λ_i is decreasing in b_i and $b_i^h > t_i$, we have the following equation:

$$(b_i^h - t_i)\lambda_i(b_{-i}, b_i^h) < \int_{t_i}^{b_i^h} \lambda_i(b_{-i}, x)dx$$

From this relation it can be seen that $P_h < P_t$. The same argument applies to the case when computer i bids lower.

Because the optimal algorithm assumes a central dispatcher, the mechanism will be implemented in a centralized way as part of the dispatcher code. We assume that the dispatcher is run on one of the computers and is able to communicate with all the other computers in the distributed system. In the following we present the protocol that implements our load balancing mechanism (LBM). This protocol has two phases: bidding and completion.

Protocol LBM:

Phase I: Bidding

1. The dispatcher sends a request for bids message (*ReqBid*) to each computer in the system.
2. When a computer receives a *ReqBid* it replies with its bid b_i to the dispatcher.

Phase II: Completion

1. After the dispatcher collects all the bids it does the following:
   - 1.1. Computes the allocation using OPTIM algorithm.
   - 1.2. Computes the payments P_i for each computer using equation (5.16).
   - 1.3. Sends P_i to each computer i.
2. Each computer receives its payment and evaluates its profit.

This protocol is executed periodically or when there is a change in the total job arrival rate. During two executions of this protocol the jobs will be allocated to computers by the dispatcher according to the allocation computed by OPTIM. Computers will receive the maximum profit only when they report the true value.

5.5 Experimental results

To study the effectiveness of our truthful mechanism we simulated a heterogeneous system consisting of 16 computers with four different processing rates. In Table 5.1 we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We choose 0.013 jobs/sec. as the processing rate for the slowest computer because it is a value that can be found in real distributed systems [127]. Also we consider only computers that are at most ten times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

Table 5.1. System configuration.

Relative processing rate	1	2	5	10
Number of computers	6	5	3	2
Processing rate (jobs/sec)	0.013	0.026	0.065	0.13

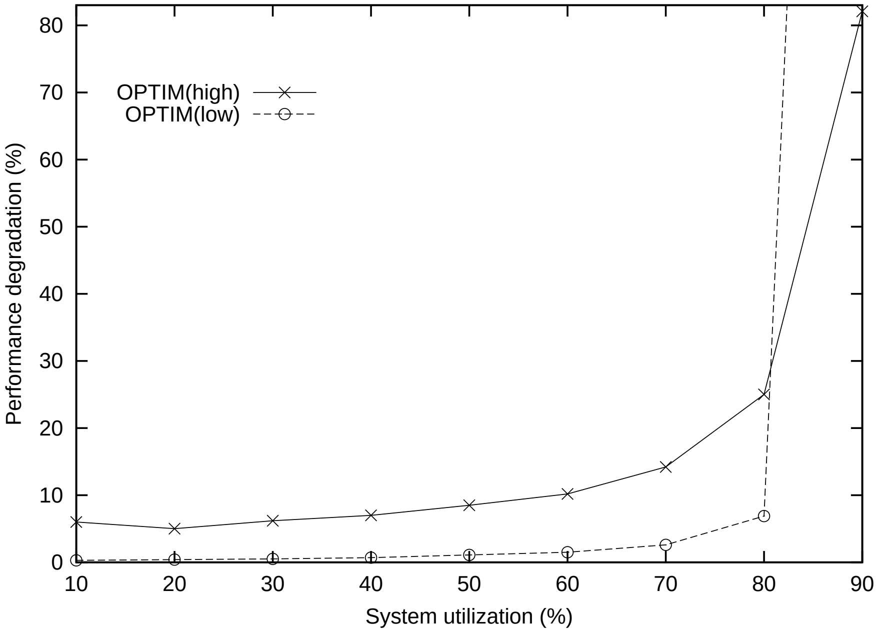

Figure 5.2. Performance degradation vs. system utilization.

We study the performance degradation due to false bids declarations. We define *performance degradation(PD)* as follows: $PD = ((D_F - D_T)/D_T)100\%$, where D_F is the response time of the system when one or more computers report false values; and D_T is the “true” optimal response time of the system when all computers report their true values. PD quantifies the increase in the execution time due to false bidding. As we pointed out in the previous section, if all the computers report their true values then the ”true” optimum is obtained and $PD = 0$. We expect an increase in the response time and in PD when one or more computers lie.

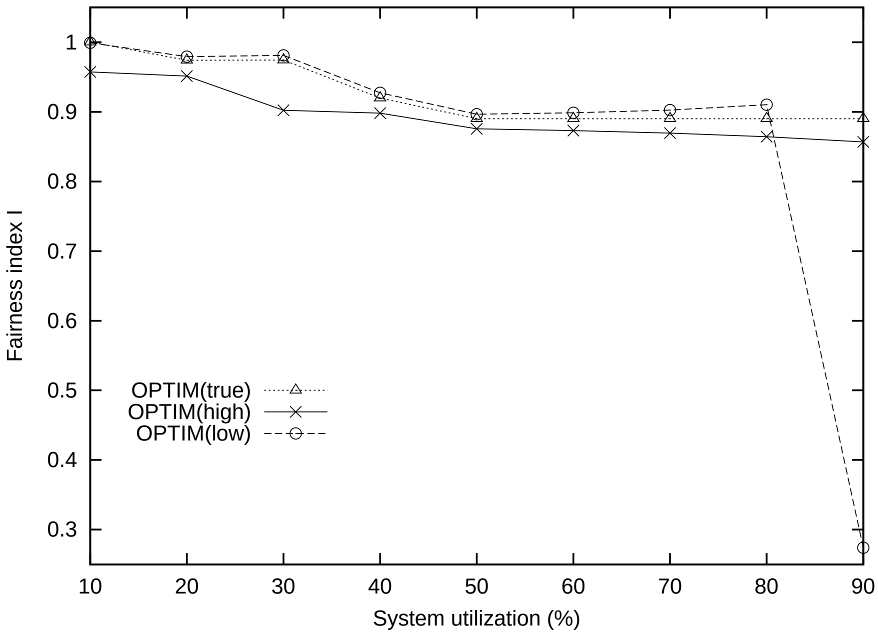

Figure 5.3. Fairness index vs. system utilization.

In our experiments we consider that the fastest computer C_1 declares false bids. C_1 has $t_1 = 1/\mu_1 = 7.69$ as its true value. In Figure 5.2 we present the degradation in expected response time of the system for different values of system utilization (ranging from 10% to 90%) and two types of bidding: overbidding and underbidding. *System utilization* (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system: $\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i}$. In the first experiment, C_1 bids 7% lower than the true value. In this case the performance degradation is around 2% for low and medium system utilization, increasing drastically (around 300%) for high system utilization. This increase is due to computer C_1 overloading. The overloading occurs because C_1 bids lower, that means it reports a higher value for its processing rate. The algorithm will allocate more jobs to C_1 increasing its response time. This increase is reflected in the expected response time of the system. In the second experiment, C_1 bids 33% higher than the true value. In this case the performance degradation is about 6% at low system utilization, about 15% at medium system utilization and more than 80% at high system utilization. It can be observed that small deviations from the true value of only one computer may lead to large values of performance degradation. If we consider that more than one computer does not report its true value then we expect very poor performance. This justifies the need for a mechanism

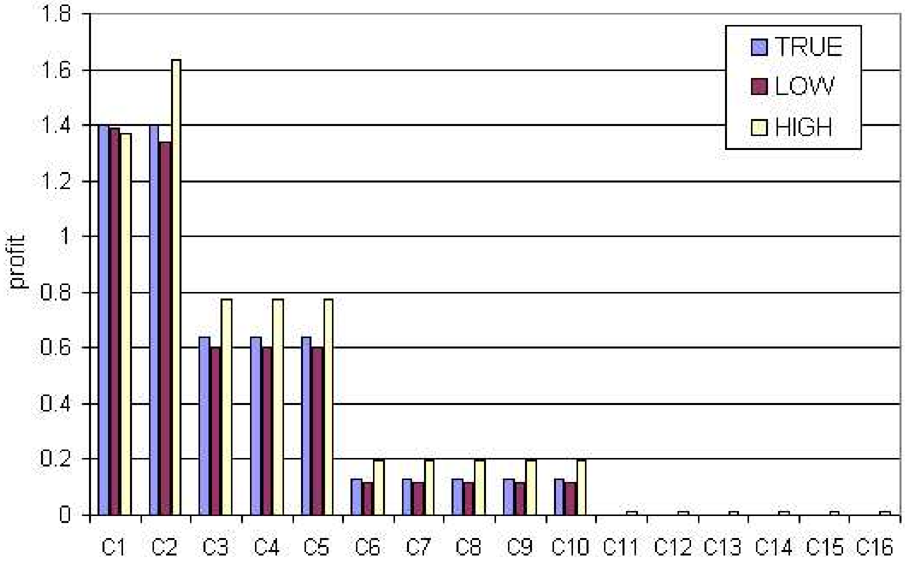

Figure 5.4. Profit for each computer (medium system load)

that will force the computers to declare their true values.

In Figure 5.3 we present the variations in the fairness index [64]. It can be observed that in the case of underbidding the fairness index decreases drastically at high system loads. This is because C_1's response time is much higher than that of the other computers. The fairness index is maintained around 90% for all other values. If more than one computer does not report its true value then we expect small variations in the fairness index. This can also be seen from the definition of this index.

In Figure 5.4 we present the profit for each computer at medium system loads (ρ = 50%). It can be observed that the profit at C_1 is maximum if it bids the true value, 3% lower if it bids higher and 1% lower if it bids lower. The mechanism penalizes C_1 if it does not report the true value. When C_1 bids lower the other computer's profits are lower because their payments decrease. Computers C_{11} to C_{16} are not utilized when C_1 underbids and when it reports the true value, thus they will not gain anything ($profit_i$ = 0, $i = 11, 12, \dots, 16$). These computers will be utilized in the case when C_1 overbids, getting a small profit. When C_1 overbids the profit for all the computers except C_1 is higher than in the case when C_1 bids the true value. This is because the payments increase for these computers.

An important issue is the *frugality* of our mechanism. We say that a mechanism is *frugal* if the mecha-

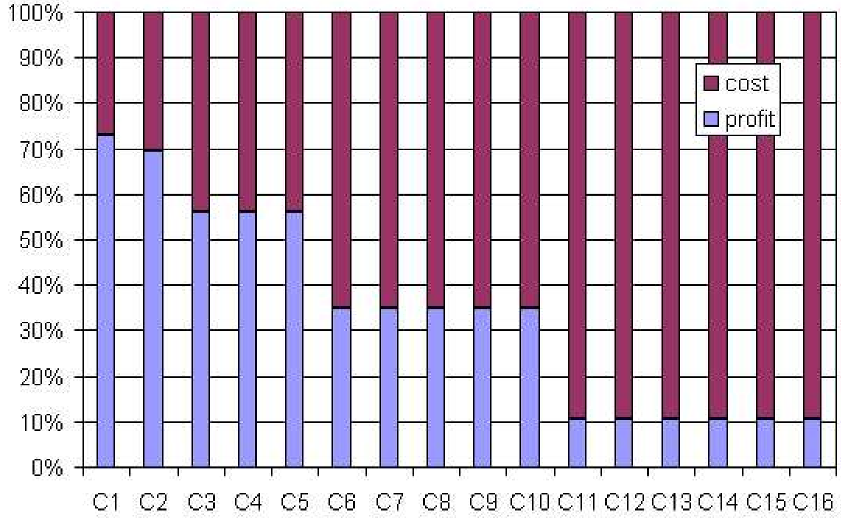

Figure 5.5. Payment structure for each computer (C_1 bids higher)

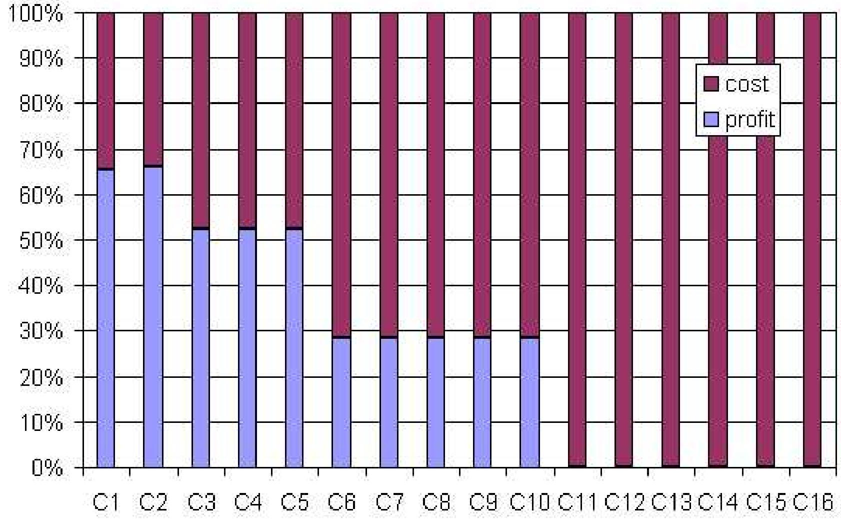

Figure 5.6. Payment structure for each computer (C_1 bids lower)

nism's payments are small by some measure [8]. This property gives us an idea of how efficient a mechanism is. The mechanism is interested in keeping its payments as small as possible. Each payment consists of an execution cost plus a profit for the computer which runs the jobs. Our mechanism must preserve voluntary participation, so the lower bound on its payments is the total cost incurred by the computers.

In Figure 5.5 and 5.6 we present the cost and profit as fractions of the payment received by each computer

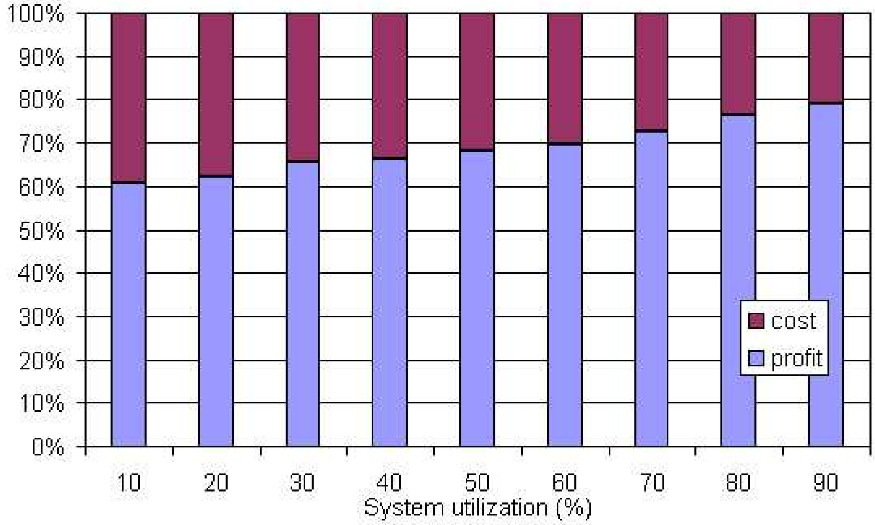

Figure 5.7. Total payment vs. system utilization.

at medium loads. In these figures we are interested to see how close to the cost is the payment to each computer. It can be observed that the cost incurred by C_1 when it bids higher is about 25% of the payment. In the case when C_1 bids lower its cost is about 35% of the payment. For the other computers the cost is between 50% and 90% when C_1 bids higher and between 53% and 100% when C_1 bids lower. For the distributed system considered in these experiments (medium loads) the highest payment given to a computer is about 3 times its cost.

In Figure 5.7 we present the total cost and profit as fractions of the total payment for different values of system utilization when C_1 reports its true value. The total cost is about 21% of the payment at 90% system utilization which is the smallest percentage. The percentage of cost increases to 40% at 10% system utilization. At medium loads the mechanism pays less than 3 times the total cost. When C_1 bids lower and higher the percentages are similar and are not presented here. We expect that these values are also valid considering other parameters of the distributed system.

5.6 Conclusion

In current distributed systems such as computational grids, resources belong to different self interested agents or organizations. These agents may manipulate the load allocation algorithm in their own benefit and their selfish behavior may lead to severe performance degradation and poor efficiency.

In this chapter we investigated the problem of designing protocols for resource allocation involving selfish agents. Solving this kind of problems is the object of mechanism design theory. Using this theory we designed a truthful mechanism for solving the load balancing problem in heterogeneous distributed systems. We proved that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. We derived a protocol that implements our mechanism and presented experiments to show its effectiveness.

Chapter 6

A Load Balancing Mechanism with Verification

6.1 Introduction

The goal of this chapter is to design a load balancing mechanism with verification. We consider a distributed system in which computers are modeled by linear load-dependent latency functions [6,118]. Solving the load balancing problem involves finding an allocation that minimizes the total latency of the system. The optimal allocation is obtained by assigning jobs to computers in proportion to their processing rates. This allocation algorithm is the basis for our mechanism. To design our mechanism we assume that each computer in the distributed system is characterized by its processing rate and that the true value of this rate is private knowledge. The *valuation* of each computer is the function that quantifies its benefit or loss and is equal to the negation of its latency.

The load balancing mechanism with verification works as follows. It first asks the computers to report their processing rates. Having obtained these rates, the mechanism computes the allocation and allocates the jobs to computers. Because we want to have a mechanism with verification the payment to each agent is computed and given to it after the assigned jobs were executed. Here we assume that the processing rate with which the jobs were actually executed is known to the mechanism. Each computer goal is to report a value for its processing rate such that its utility is maximized. The *utility* of each computer is defined as the sum of its valuation and its payment. The mechanism must be designed such that the agents maximize their utility only if they report the true values of the processing rates and execute the jobs using their full processing capacity. Also, if each computer reports its true value and executes the jobs at the full capacity then the minimum total latency is obtained.

In this chapter we design a mechanism with verification based on a compensation and bonus type mechanism. This type of mechanism was initially studied by Nisan and Ronen [108] in the context of task scheduling on unrelated machines.

Our contributions

In this chapter we design a load balancing mechanism with verification in heterogeneous distributed systems. We model the computers of the distributed system using linear load-dependent latency functions. We formulate the problem of designing a load balancing mechanism with verification and we devise a truthful mechanism for this problem. We prove that our mechanism is truthful and satisfies the voluntary participation condition. A simulation study is performed to investigate the effectiveness of our mechanism.

Organization

The chapter is structured as follows. In Section 6.2 we present the distributed system model and formulate the load balancing problem. In Section 6.3 we design a truthful mechanism with verification that solves the load balancing problem in distributed systems in which computers have linear load-dependent latency functions. In Section 6.4 we investigate the effectiveness of our load balancing mechanism by simulation. In Section 6.5 we draw conclusions.

6.2 Model and problem formulation

We consider a distributed system that consists of a set $N = \{1, 2, \ldots, n\}$ of n heterogeneous computers. We assume that each computer is characterized by a load-dependent *latency function*. The latency function of computer i is linear on x_i and has the following form:

$$l_i(x_i) = a_i x_i \tag{6.1}$$

where x_i is the arrival rate of jobs allocated to computer i and a_i is a parameter inversely proportional to the processing rate of computer i. A small (big) a_i characterizes a fast (slow) computer. In other words $l_i(x_i)$ measures the time required to complete one job on computer i.

Models using linear load dependent latency functions were investigated in [6,118]. This type of function could represent the expected waiting time in a M/G/1 queue, under light load conditions (considering a_i as the variance of the service time in the queue) [6].

We assume that there is a large number of jobs that need to be executed. These jobs arrive at the system with an arrival rate R. A feasible allocation of jobs to the computers is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that satisfies two conditions:

(i) Positivity: $x_i > 0 \qquad i = 1, 2, \ldots, n$;

(ii) Conservation: $\sum_{i=1}^{n} x_i = R$;

The performance of the system is characterized by the *total latency*:

$$L(\mathbf{x}) = \sum_{i=1}^{n} x_i l_i(x_i) = \sum_{i=1}^{n} a_i x_i^2 \tag{6.2}$$

The load balancing problem can be stated as follows:

Definition 6.1 (Load balancing problem) Given the job arrival rate R at a distributed system with n heterogeneous computers characterized by the load-dependent latency functions $l_i(.)$, find a feasible allocation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that minimizes the total latency $L(\mathbf{x})$.

The following theorem gives the solution for this problem.

Theorem 6.1 (Optimal allocation) The optimal allocation for the above load balancing problem is given by:

$$x_i = \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}} R \qquad i = 1, 2, \ldots, n \tag{6.3}$$

This allocation gives the minimum value for the total latency:

$$L^* = \frac{R^2}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{6.4}$$

Proof: In Appendix D.

In other words this theorem says that if the latency functions for each computer are linear then the allocation of jobs in proportion to the processing rate of each computer gives the minimum total latency. Using this theorem, the following algorithm can be derived.

PR algorithm:

```
Input: Processing rates: $\frac{1}{a_1}, \frac{1}{a_2}, \dots \frac{1}{a_n}$;
       Arrival rate: $R$;
Output: Load allocation: $x_1, x_2, \dots x_n$;
       for $i = 1, \dots, n$ do
           $x_i \leftarrow R \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}}$;
```

The above algorithm solves the load balancing problem in the classical setting where the participants (computers) are assumed to follow the algorithm. Here we assume that computers are selfish agents characterized by their true values t_i, $i = 1, 2, \dots, n$. The true value t_i corresponds to the parameter a_i in the latency function l_i, which is inversely proportional to the processing rate of computer i. We need to design a mechanism that obtains the optimal allocation and forces the participants to reveal their true types t_i and follow the PR algorithm. In the next section we design such a mechanism.

6.3 The load balancing mechanism with verification

In this section we formally describe the load balancing mechanism design problem considering our distributed system model. Then we present the design of our load balancing mechanism with verification and study its properties.

Definition 6.2 (Mechanism design problem) The problem of designing a load balancing mechanism with verification is characterized by:

(i) A finite set $\mathbf{X}$ of allowed outputs. The output is a vector $\mathbf{x}(\mathbf{b}) = (x_1(\mathbf{b}), x_2(\mathbf{b}), \dots, x_n(\mathbf{b}))$, $\mathbf{x}(\mathbf{b}) \in \mathbf{X}$, computed according to the agents' bids, $\mathbf{b} = (b_1, b_2, \dots, b_n)$. Here, b_i is the value (bid) reported by agent i to the mechanism.

(ii) Each agent i, ($i = 1, \dots, n$), has a privately known parameter t_i called its *true value* and a publicly known parameter $\tilde{t}_i \geq t_i$ called its *execution value*. The preferences of agent i are given by a function called *valuation* $V_i(\mathbf{x}, \tilde{\mathbf{t}}) = -\tilde{t}_i x_i^2$. The execution value $\tilde{t}_i$ determines the value of the latency function and thus the actual execution time for one job at agent i.

(iii) Each agent goal is to maximize its *utility*. The utility of agent i is $U_i(\mathbf{b}, \tilde{\mathbf{t}}) = P_i(\mathbf{b}, \tilde{\mathbf{t}}) + V_i(\mathbf{x}(\mathbf{b}), \tilde{\mathbf{t}})$, where P_i is the payment handed by the mechanism to agent i and $\tilde{\mathbf{t}}$ is the vector of execution values. The payments are handed to agents after the assigned jobs have been completed and the mechanism knows $\tilde{t}_i$, $i = 1, 2, \dots, n$.

(iv) The goal of the mechanism is to select an output $\mathbf{x}$ that minimizes the total latency function $L(\mathbf{x}, \mathbf{b}) = \sum_{i=1}^{n} b_i x_i^2$.

An agent i may report a value (bid) b_i different from its true value t_i. The true value characterizes the actual processing capacity of computer i. In addition, agent i may choose to execute the jobs allocated to it with a different processing rate given by its execution value ($\tilde{t}_i \geq t_i$). Thus, an agent i may execute the assigned jobs at a slower rate than its true processing rate. The goal of a truthful mechanism with verification is to give incentives to agents such that it is beneficial for them to report their true values and execute the assigned jobs using their full processing capacity. Now we give a formal description of a mechanism with verification.

Definition 6.3 (Mechanism with verification) A *mechanism with verification* is a pair of functions:

(i) The *allocation function* $\mathbf{x}(\mathbf{b}) = (x_1(\mathbf{b}), x_2(\mathbf{b}), \dots, x_n(\mathbf{b}))$. This function has as input the vector of agents' bids $\mathbf{b} = (b_1, b_2, \dots, b_n)$ and returns an output $\mathbf{x} \in \mathbf{X}$.

(ii) The *payment function* $P(\mathbf{b}, \tilde{\mathbf{t}}) = (P_1(\mathbf{b}, \tilde{\mathbf{t}}), P_2(\mathbf{b}, \tilde{\mathbf{t}}), \dots, P_n(\mathbf{b}, \tilde{\mathbf{t}}))$, where $P_i(\mathbf{b}, \tilde{\mathbf{t}})$ is the payment handed by the mechanism to agent i.

According to this definition, to obtain a load balancing mechanism with verification we must find an allocation function $\mathbf{x}(\mathbf{b})$ and a payment function $P(\mathbf{b}, \tilde{\mathbf{t}})$. The allocation function must minimize $L(\mathbf{x})$. The payment function must be designed such that the mechanism is truthful.

Here we consider a compensation and bonus type mechanism [108] to solve the load balancing problem. The optimal allocation function for this mechanism is given by the PR algorithm. The payment function for this type of mechanism is the sum of two functions: a compensation function and a bonus function.

In the following we define our load balancing mechanism with verification.

Definition 6.4 (The load balancing mechanism with verification) The mechanism with verification that solves the load balancing problem is defined by the following two functions:

(i) The allocation function given by the PR algorithm.

(ii) The payment function is given by:

$$P_i(\mathbf{b}, \tilde{\mathbf{t}}) = C_i(\mathbf{b}, \tilde{\mathbf{t}}) + B_i(\mathbf{b}, \tilde{\mathbf{t}}) \tag{6.5}$$

where the function $C_i(\mathbf{b}, \tilde{\mathbf{t}}) = \tilde{t}_i x_i^2(\mathbf{b})$ is called the *compensation* function for agent i; and the function $B_i(\mathbf{b}, \tilde{\mathbf{t}}) = L_{-i}(\mathbf{x}(\mathbf{b}_{-i}, \mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i))$ is called the *bonus* for agent i. The function $L_{-i}(\mathbf{x}(\mathbf{b}_{-i}, \mathbf{b}_{-i}))$ is the optimal latency when agent i is not used in the allocation. Thus, the bonus for an agent is equal to its contribution in reducing the total latency.

We are interested in obtaining a truthful load balancing mechanism. For our load balancing mechanism we can state the following theorem.

Theorem 6.2 (Truthfulness) The load balancing mechanism with verification is truthful.

Proof: In Appendix D.

A desirable property of a mechanism is that the profit of a truthful agent is always non-negative. This means the agents hope for a profit by participating in the mechanism.

Definition 6.5 (Voluntary participation mechanism) We say that a mechanism with verification satisfies the *voluntary participation condition* if $U_i((\mathbf{b}_{-i}, t_i), \tilde{t}_i) \geq 0$ for every agent i, true values t_i, execution values $\tilde{t}_i = t_i$, and other agents' bids $\mathbf{b}_{-i}$ (i.e. truthful agents never incur a loss).

Theorem 6.3 (Voluntary participation) The load balancing mechanism with verification satisfies the voluntary participation condition.

Proof: In Appendix D.

We obtained a truthful load balancing mechanism with verification that satisfies the voluntary participation condition. Based on this mechanism a load balancing protocol can be derived. An informal description of this centralized protocol is as follows. The mechanism collects the bids from each computer, computes the allocation using PR algorithm and allocates the jobs. Then it waits for the allocated jobs to be executed. In this waiting period the mechanism estimates the actual job processing rate at each computer and use it to determine the execution value $\tilde{t}_i$. After the allocated jobs are completed the mechanism computes the payments and sends them to the computers. After receiving the payment each computer evaluates its utility.

6.4 Experimental results

In this section we study the effectiveness of the proposed mechanism by simulation. The simulated distributed system consists of 16 heterogeneous computers. The latency function of each computer is given by the parameter $a_i = t_i$ presented in Table 6.1. This parameter is inversely proportional to the computer's processing rate.

Table 6.1. System configuration.

Computers	C1 - C2	C3 - C5	C6 - C10	C11 - C16
True value (t)	1	2	5	10

We consider eight types of experiments depending on the bid and on the execution value of computer C1. In all the experiments we assume that all the computers except C1 bid their true values and that their execution values are the same as their true values. We divide these experiments in three main classes according to the bids of computer C1 as follows: *True*, when $b_1 = t_1$; *High*, when $b_1 > t_1$; and *Low*, when $b_1 < t_1$. We further divide these main classes considering the execution value of C1. We have two sets of experiments for the *True* class, four for the *High* class and two for the *Low* class. The parameters used in these experiments are presented in Table 6.2.

Table 6.2. Types of experiments.

Experiment	Characterization	t_1	b_1	$\tilde{t}_1$
True1	$\tilde{t}_1 = t_1 = b_1$	1	1	1
True2	$\tilde{t}_1 > t_1 = b_1$	1	1	3
High1	$\tilde{t}_1 = b_1 > t_1$	1	3	3
High2	$b_1 > t_1 = \tilde{t}_1$	1	3	1
High3	$b_1 > \tilde{t}_1 > t_1$	1	3	2
High4	$\tilde{t}_1 > b_1 > t_1$	1	3	4
Low1	$\tilde{t}_1 = t_1 > b_1$	1	0.5	1
Low2	$\tilde{t}_1 > t_1 > b_1$	1	0.5	2

First, we consider the influence of false bids ($b_1 \neq t_1$) on the total latency. We assume that the job rate is $R = 20$ jobs/sec. In Figure 6.1 we present the total latency for the eight experiments. We now discuss the results of each experiment.

***True1*:** All the computers report their true values. The execution value is equal to the true value for each computer. As expected (from the theory) we obtain the minimum value for the total latency (L = 78.43).

***True2*:** The parameters are as in *True1* except that C1 has a higher execution value $\tilde{t}_1 > t_1$. This means C1 execution is slower increasing the total latency by 17%.

***High1*:** In this case C1 bids three times higher than its true value and the execution value is equal to the bid. Because C1 bids higher the other computers are overloaded thus increasing the total latency. C1 gets fewer jobs and executes them with a slower execution rate.

***High2*:** In this case C1 gets fewer jobs and the other computers are overloaded. Here the increase is not as big as in *High1* because C1 executes the jobs at its full capacity.

***High3*:** This case is similar to *High1* except that the execution on C1 is faster. It can be seen that the total latency is less than in the case *High1*.

***High4*:** Similar to *High1*, except that C1 executes the jobs slower, increasing the total latency.

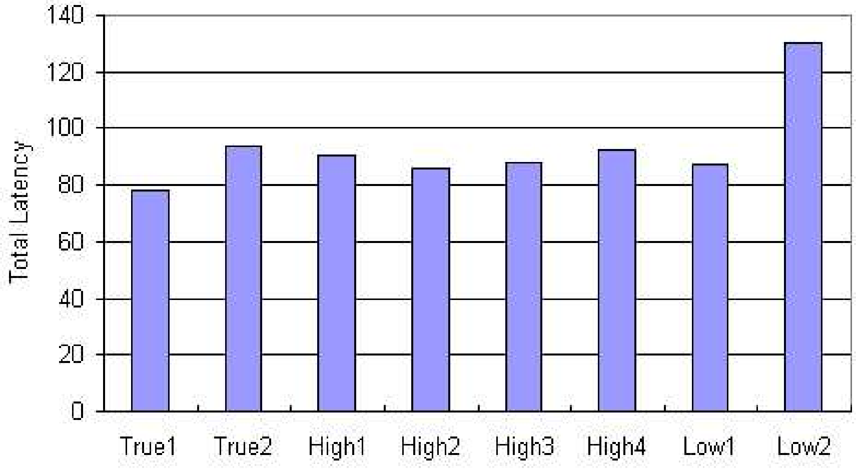

Figure 6.1. Total latency for each experiment.

***Low1*:** C1 bids 2 times less than its true value, getting more jobs and executing them at its full capacity. The increase in total latency is not big, about 11%.

***Low2*:** In this case C1 gets more jobs and executes them two times slower. This overloading of C1 leads to a significant increase in the total latency (about 66%).

From the results of these experiments, it can be observed that small deviations from the true value and from the execution value of only one computer may lead to large values of the total latency. We expect even larger increase if more than one computer does not report its true value and does not use its full processing capacity. The necessity of a mechanism that forces the participants to be truthful becomes vital in such situations.

In Figure 6.2 we present the payment and utility of computer C1 for each set of experiments. As expected, C1 obtains the highest utility in the experiment *True1*, when it bids its real value and uses its full processing capacity. In the other experiments C1 is penalized for lying and the payment that it receives is lower than in the case of *True1*. The utility is also lower in these experiments. An interesting situation occurs in the experiment *Low2* where the payment and utility of C1 are negative. This can be explained as follows. Computer C1 bids two times less than its true value and the PR algorithm allocates more jobs to it. In addition, its execution value $\tilde{t}_1$ is two times higher than t_1, that means the allocated jobs are executed two

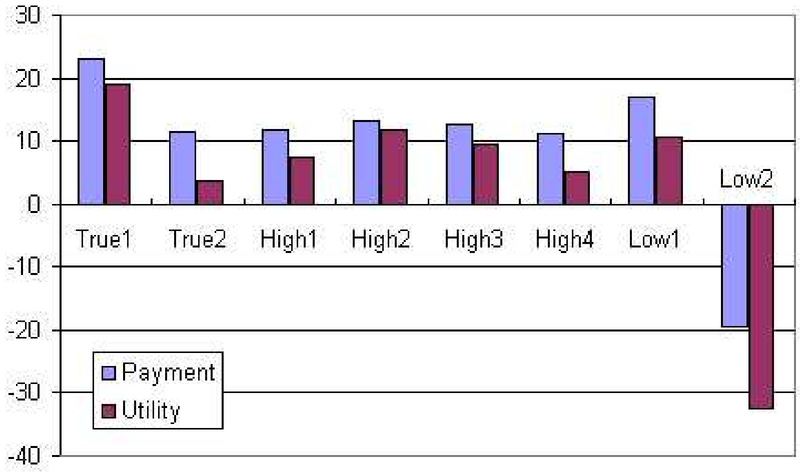

Figure 6.2. Payment and utility for computer C1.

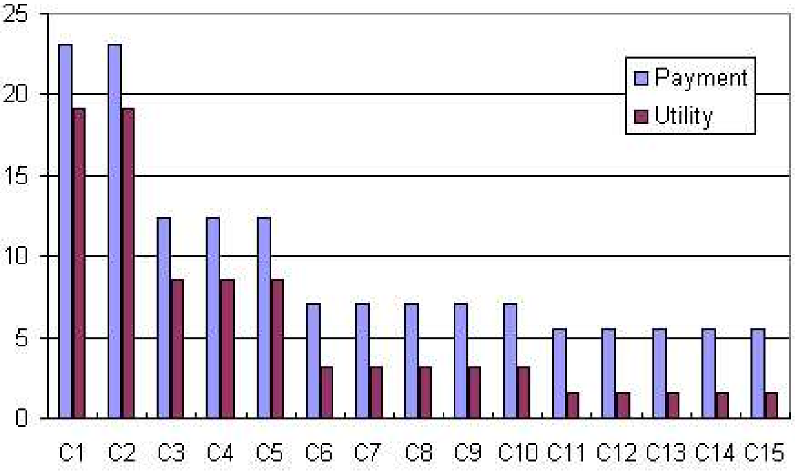

Figure 6.3. Payment and utility for each computer (*True1*).

times slower than in the experiment *True1*(when $\tilde{t}_1 = t_1$). More allocated jobs to C1 combined with a slow execution increases the total latency L. The total latency becomes greater than the latency L_{-1} obtained when computer C1 is not used in the allocation (i.e. $L > L_{-1}$) and thus the bonus is negative. The absolute value of the bonus is greater than the compensation and from the definition of the payment it can be seen that the payment given to C1 is negative.

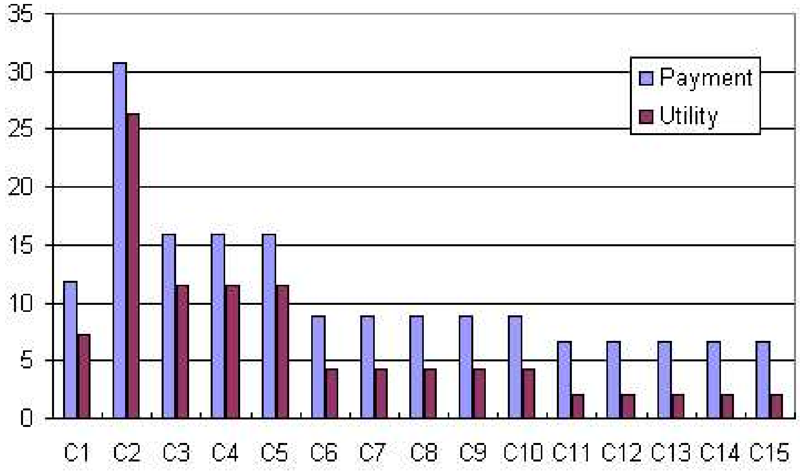

Figure 6.4. Payment and utility for each computer (*High1*).

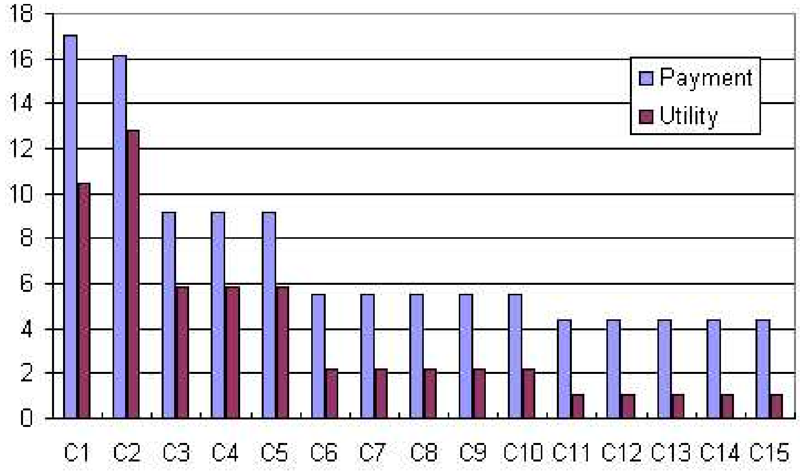

Figure 6.5. Payment and utility for each computer (*Low1*).

In Figures 6.3-6.5 we present the payment structure for each computer in three of the experiments. In the experiment *Low1* (Figure 6.5) computer C1 obtains a utility which is 45% lower than in the experiment *True1*. The other computers (C2 - C16) obtain lower utilities. This is because all the computers except C1 receives fewer jobs and their payments are smaller than in the experiment *True1*.

In the experiment *High1*(Figure 6.4) computer C1 obtains a utility which is 62% lower than in the

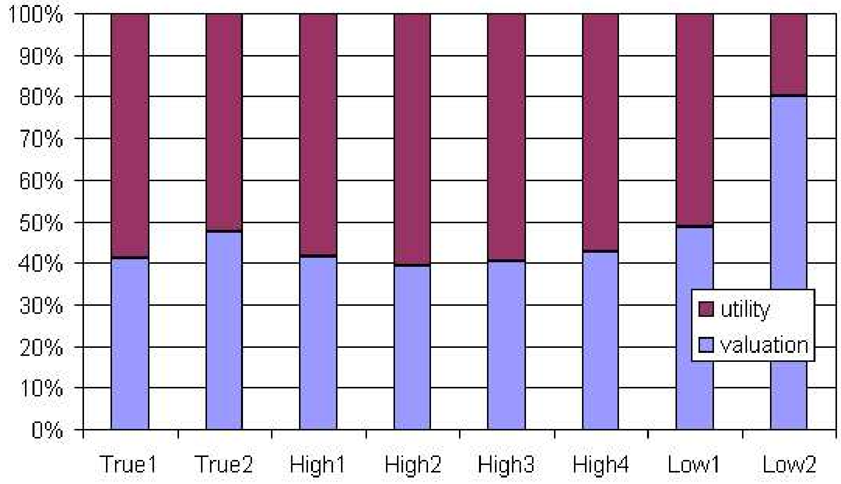

Figure 6.6. Payment structure.

experiment *True1*. The other computers (C2 - C16) obtain higher utilities. The mechanism pays them more because they receive more jobs.

In Figure 6.6 we present the payment structure of our mechanism. It can be observed that the total payment given by our mechanism to the computers is at most 2.5 times the total valuation. The lower bound on the total payment is the total valuation. This is because our mechanism must preserve the voluntary participation property. The amount of the total payment given by a mechanism characterizes its frugality. A mechanism that gives small payments by some measure it is called frugal. Based on the experimental results presented here the payments are at most 2.5 times the total valuation, thus our mechanism can be considered frugal.

6.5 Conclusion

We designed a truthful mechanism with verification that solves the load balancing problem in a distributed system in which computers are characterized by linear load-dependent latency functions. We proved that this mechanism satisfies the voluntary participation condition. We studied the performance of this mechanism in terms of its payment structure.

Chapter 7

Conclusions

In this chapter, we summarize our work and describe the contributions of this dissertation. We begin with a review of our results on load balancing games. Next, we discuss our results on algorithmic mechanism design for load balancing. Finally, we conclude by presenting some possible directions for future research.

7.1 Load balancing games

Most of the previous work on static load balancing considered as their main objective the minimization of overall expected response time. The fairness of allocation, which is an important issue for modern distributed systems, has received relatively little attention. We developed a formal framework, based on cooperative game theory for characterization of fair allocation schemes that are also optimal for each job. Using this framework we formulated the load balancing problem in single class job distributed systems as a cooperative game among computers. We showed that the *Nash Bargaining Solution*(NBS) provides a Pareto optimal operation point for the distributed system. We gave a characterization of NBS and an algorithm for computing it. We proved that NBS provides a fair allocation of jobs to computers. We conducted a simulation study in order to compare its performance with other existing load balancing algorithms.

The minimization of overall expected response time is difficult to achieve in distributed systems where there is no central authority controlling the allocation and users are free to act in a selfish manner. We formulated the load balancing problem in distributed systems as a noncooperative game among users. The Nash equilibrium provides a user-optimal operation point for the distributed system and represents the solution of our noncooperative load balancing game. We gave a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compared the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system.

7.2 Algorithmic mechanism design for load balancing

We designed a truthful mechanism for solving the static load balancing problem in heterogeneous distributed systems. We proved that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. Based on the proposed mechanism we derived a load balancing protocol and we studied its effectiveness by simulations.

We studied the problem of designing load balancing mechanisms with verification. An agent may report a value (bid) different from its true value. Here the true value characterizes the actual processing capacity of each computer. In addition, an agent may choose to execute the jobs allocated to it with a different processing rate given by its execution value. Thus, an agent may execute the assigned jobs at a slower rate than its true processing rate. The goal of a truthful mechanism with verification is to give incentives to agents such that it is beneficial for them to report their true values and execute the assigned jobs using their full processing capacity. This assumes that the mechanism knows the execution values after the jobs were completed. We designed a truthful mechanism with verification that solves the load balancing problem and satisfies the voluntary participation condition. We studied the effectiveness of our load balancing mechanism by simulation.

7.3 Future research directions

Noncooperative load balancing based on bayesian games

The main assumption in the previous work on load balancing noncooperative games is that users have complete information (i.e. each user knows the payoffs of the others). In real distributed systems this may not be a reasonable assumption due to possible limited information, uncertainty on system's parameters and influence of external random events. In the future we would like to develop a model for load balancing under uncertainty based on Bayesian noncooperative games. Several problems arise considering this model: proving the existence of the Bayesian-Nash equilibrium, providing an algorithm for computing the equilibrium and studying its complexity.

Fault tolerant mechanism design for resource allocation

One interesting extension of our work on mechanism design for load balancing is to consider that each agent (computer) is characterized not only by its processing rate, but also by its probability of failure. We would like to study this problem and devise a fault tolerant load balancing mechanisms that exhibits good properties such as truthfulness and voluntary participation.

Agents' privacy in mechanism design

We would like to investigate the issue of designing distributed load balancing mechanisms in which the agents' parameters remain private knowledge. One idea is to use cryptographic protocols for Secure Multiparty Function Evaluation (SMFE) for computing the allocations and payments. This protocols cannot be directly applied because they assume a different strategic model and they have high network complexity. We would like to modify these protocols such that they can be used in designing distributed load balancing mechanisms.

Appendix A

Proofs from Chapter 3

A.1 Proof of Theorem 3.4

In Theorem 3.1 we consider $f_i(\beta_i) = -\beta_i$ which are concave and bounded above. The set defined by the constraints is convex and compact. The condition of 3.1 are satisfied and the result follows. □

A.2 Proof of Theorem 3.5

Using the fact that $f_i(\beta_i) = -\beta_i$ are one-to-one functions of β_i and applying Theorem 3.3 the results follows. □

A.3 Proof of Theorem 3.6

The constraints in Theorem 3.5 are linear in β_i and $f_i(\beta_i) = -\beta_i$ has continuous first partial derivatives. This imply that the first order Kuhn-Tucker conditions are necessary and sufficient for optimality [94].

Let $\alpha \leq 0$, $\eta_i \leq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [94]. The Lagrangian is:

$$L(\beta_i, \alpha, \eta_i) = \sum_{i=1}^{n} \ln(\mu_i - \beta_i) + \alpha(\sum_{i=1}^{n} \beta_i - \Phi) + \sum_{i=1}^{n} \eta_i(\beta_i - \mu_i) \tag{A.1}$$

The first order Kuhn-Tucker conditions are:

$$\frac{\partial L}{\partial \beta_i} = -\frac{1}{\mu_i - \beta_i} + \alpha + \eta_i = 0, \qquad i = 1, \ldots, n \tag{A.2}$$

$$\frac{\partial L}{\partial \alpha} = \sum_{i=1}^{n} \beta_i - \Phi = 0 \tag{A.3}$$

$$\mu_i - \beta_i \geq 0, \quad \eta_i(\mu_i - \beta_i) = 0, \quad \eta_i \leq 0, \qquad i = 1, \ldots, n \tag{A.4}$$

We have $\mu_i - \beta_i > 0, \quad i = 1, \ldots, n$. This implies that $\eta_i = 0, \quad i = 1, \ldots, n$.

The solution of these conditions is:

$$\frac{1}{\mu_i - \beta_i} - \alpha = 0, \qquad i = 1, \ldots, n \tag{A.5}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{A.6}$$

It then follows that:

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n} \tag{A.7}$$

□

Lemma A.1 Let $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_m$. If $\mu_m < \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}$ then the objective function from Theorem 3.5 is maximized, subject to the extra constraint $\beta_m \geq 0$, when $\beta_m = 0$. □

Proof: We add a new condition to the first order conditions (A.2-A.4). The new set of conditions is as follows:

$$\frac{\partial L}{\partial \beta_i} = -\frac{1}{\mu_i - \beta_i} + \alpha + \eta_i = 0, \qquad i = 1, \ldots, m-1 \tag{A.8}$$

$$\frac{\partial L}{\partial \beta_m} = -\frac{1}{\mu_m - \beta_m} + \alpha + \eta_m + \gamma_m = 0 \tag{A.9}$$

$$\frac{\partial L}{\partial \alpha} = \sum_{i=1}^{m} \beta_i - \Phi = 0 \tag{A.10}$$

$$\mu_i - \beta_i \geq 0, \quad \eta_i(\mu_i - \beta_i) = 0, \quad \eta_i \leq 0, \qquad i = 1, \ldots, m \tag{A.11}$$

$$\beta_m \geq 0, \quad \gamma_m \beta_m = 0, \quad \gamma_m \leq 0 \tag{A.12}$$

We have $\mu_i - \beta_i > 0, \quad i = 1, \ldots, m$. This implies that $\eta_i = 0, \quad i = 1, \ldots, m$.

$$\frac{1}{\mu_i - \beta_i} - \alpha = 0, \qquad i = 1, \ldots, m-1 \tag{A.13}$$

$$\frac{1}{\mu_m - \beta_m} - \alpha + \gamma_m = 0 \tag{A.14}$$

$$\sum_{i=1}^{m} \beta_i = \Phi \tag{A.15}$$

$$\beta_m \geq 0, \quad \gamma_m \beta_m = 0, \quad \gamma_m \leq 0 \tag{A.16}$$

From equation (A.16) the value of β_i is either zero or positive. We consider each case separately.

Case 1: $\beta_m > 0$. Equation (A.16) implies $\gamma_m = 0$ and the solution of the conditions is:

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m} \quad i = 1, \ldots, m \tag{A.17}$$

Case 2: $\beta_m = 0$. It follows from equation (A.16) that $\gamma_m \leq 0$.

The restriction $\beta_m \geq 0$ becomes active when $\frac{1}{\mu_m - \beta_m} - \alpha = -\gamma_m > 0$ which implies that $\mu_m < \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}$ and lemma is proved. □

A.4 Proof of Theorem 3.7

The while loop in step 3 finds the minimum index m for which $\mu_m < \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}$.

If $m = n$ (that means that all β_i are positive) we apply Theorem 3.6 and the result follows.

If $m < n$ we have $\mu_i < \frac{\sum_{j=1}^{i} \mu_j - \Phi}{i}$ for $i = m, \ldots, n$. According to Lemma A.1 $\beta_m, \ldots, \beta_n$ must be 0 in order to maximize the objective function from Theorem 3.5. The algorithm sets $\beta_i = 0$ for $i = m, \ldots, n$ in the while loop.

Because $\mu_m < \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}$ implies $0 < \mu_m m \leq \sum_{j=1}^{m} \mu_j - \Phi$ and then $\Phi < \sum_{j=1}^{m} \mu_j$, we can apply Theorem 3.6 to the first m indices (that corresponds to the first m fast computers) and the values of the load allocation $\{\beta_1, \beta_2, \ldots, \beta_m\}$ maximizes the objective function and are given by:

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}, \quad i = 1, \ldots, m \tag{A.18}$$

This is done in step 4. All these β_i, for $i = 1, \ldots, m$ are guaranteed to be nonnegative because $\mu_i > \frac{\sum_{j=1}^{m} \mu_j - \Phi}{m}$. The load allocation $\{\beta_1, \beta_2, \ldots, \beta_m\}$ is the solution of the optimization problem in Theorem 3.5. According to Theorem 3.5 this is also the NBS of our cooperative load balancing game. □

A.5 Proof of Theorem 3.8

Using Theorem 3.6, T_i for all n_a allocated computers ($\beta_i \neq 0$, $i = 1, \ldots, n_a$) can be expressed as:

$$T_i = \frac{1}{(\mu_i - \beta_i)} = \frac{1}{\mu_i - \mu_i + \frac{\sum_{j=1}^{n_a} \mu_j - \Phi}{n_a}} = \frac{n_a}{\sum_{j=1}^{n_a} \mu_j - \Phi} \tag{A.19}$$

Thus all T_i, $i = 1 \ldots, n_a$ are equal and this implies $I(\mathbf{T}) = 1$. □

Appendix B

Proofs from Chapter 4

B.1 Proof of Theorem 4.1

We begin with the observation that at the Nash equilibrium the stability condition (4.7) is always satisfied because of (4.3) and the fact that the total arrival rate (Φ) does not exceed the total processing rate of the distributed system. Thus we consider $\mathrm{OP_j}$ problem with only two restrictions, (4.5) and (4.6).

We first show that $D_j(\mathbf{s})$ is a convex function in $\mathbf{s}_j$ and that the set of feasible solutions defined by the constraints (4.5) and (4.6) is convex.

From (4.2) it can be easily show that $\frac{\partial D_j(\mathbf{s})}{\partial s_{ji}} \geq 0$ and $\frac{\partial^2 D_j(\mathbf{s})}{\partial (s_{ji})^2} \geq 0$ for $i = 1, \ldots, n$. This means that the Hessian of $D_j(\mathbf{s})$ is positive which implies that $D_j(\mathbf{s})$ is a convex function of the load fractions $\mathbf{s}_j$. The constraints are all linear and they define a convex polyhedron.

Thus, $\mathrm{OP_j}$ involves minimizing a convex function over a convex feasible region and the first order Kuhn-Tucker conditions are necessary and sufficient for optimality [94].

Let $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [94]. The Lagrangian is:

$$L(s_{j1}, \ldots, s_{jn}, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i^j - s_{ji}\phi_j} - \alpha(\sum_{i=1}^{n} s_{ji} - 1) - \sum_{i=1}^{n} \eta_i s_{ji} \tag{B.1}$$

The Kuhn-Tucker conditions imply that s_{ji}, $i = 1, \ldots, n$ is the optimal solution to $\mathrm{OP_j}$ if and only if there exists $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ such that:

$$\frac{\partial L}{\partial s_{ji}} = 0 \tag{B.2}$$

$$\frac{\partial L}{\partial \alpha} = 0 \tag{B.3}$$

$$\eta_i s_{ji} = 0, \quad \eta_i \geq 0, \quad s_{ji} \geq 0, \qquad i = 1, \ldots, n \tag{B.4}$$

These conditions become:

$$\frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2} - \alpha - \eta_i = 0, \qquad i = 1, \ldots, n \tag{B.5}$$

$$\sum_{i=1}^{n} s_{ji} = 1 \tag{B.6}$$

$$\eta_i s_{ji} = 0, \quad \eta_i \geq 0, \quad s_{ji} \geq 0, \qquad i = 1, \ldots, n \tag{B.7}$$

These are equivalent to:

$$\alpha = \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if} \quad s_{ji} > 0 \qquad 1 \leq i \leq n \tag{B.8}$$

$$\alpha \leq \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if} \quad s_{ji} = 0 \qquad 1 \leq i \leq n \tag{B.9}$$

$$\sum_{i=1}^{n} s_{ji} = 1, \quad s_{ji} \geq 0, \quad i = 1, \ldots, n \tag{B.10}$$

Claim: Obviously, a computer with a higher average processing rate should have a higher fraction of jobs assigned to it. Under the assumption on the ordering of computers ($\mu_1^j \geq \mu_2^j \geq \ldots \geq \mu_n^j$), we have the following order on load fractions: $s_{j1} \geq s_{j2} \geq \ldots \geq s_{jn}$. This implies that may exist situations in which the slow computers have no jobs assigned to them. This means that there exist an index c_j ($1 \leq c_j \leq n$) so that $s_{ji} = 0$ for $i = c_j, \ldots, n$.

From (B.8) and based on the above claims we can obtain by summation the following equation:

$$\sum_{i=1}^{c_j-1} \sqrt{\mu_i^j} = \sqrt{\alpha}\left(\sum_{i=1}^{c_j-1} \mu_i^j - \sum_{i=1}^{c_j-1} s_{ji}\phi_j\right) \tag{B.11}$$

Using (B.9) the above equation becomes:

$$\sqrt{\alpha} = \frac{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1}\mu_i^j - \sum_{i=1}^{c_j-1}s_{ji}\phi_j} \leq \frac{1}{\sqrt{\mu_i^{c_j}}} \tag{B.12}$$

This is equivalent to:

$$\sqrt{\mu_i^{c_j}}\sum_{i=1}^{c_j}\sqrt{\mu_i^j} \leq \sum_{i=1}^{c_j}\mu_i^j - \phi_j \tag{B.13}$$

Thus, the index c_j is the minimum index that satisfies the above equation and the result follows. □

B.2 Proof of Theorem 4.2

The while loop in step 3 finds the minimum index c_j for which $\sqrt{\mu_{c_j}^j} \leq \frac{\sum_{k=1}^{c_j}\mu_k^j-\phi_j}{\sum_{k=1}^{c_j}\sqrt{\mu_k^j}}$. In the same loop, s_{ji} are set to zero for $i = c_j, \ldots, n$. In step 4, s_{ji} is set equal to $\frac{1}{\phi_j}\left(\mu_i^j - \sqrt{\mu_i^j}\frac{\sum_{i=1}^{c_j}\mu_i^j-\phi_j}{\sum_{i=1}^{c_j}\sqrt{\mu_i^j}}\right)$ for $i = 1, \ldots, c_j - 1$. These are in accordance with Theorem 4.2. Thus, the allocation $\{s_{j1}, \ldots, s_{jn}\}$ computed by the OPTIMAL algorithm is the optimal solution of $\mathrm{OP_j}$. □

Appendix C

Proofs from Chapter 5

C.1 Proof of Theorem 5.1

We fix the other bids and consider $\lambda_i(b_{-i}, b_i)$ as a single variable function of b_i. Let $\tilde{b}_i$ and b_i be any two bids such that $\tilde{b}_i > b_i$. In terms of processing rates we have $\frac{1}{\tilde{\mu}_i} > \frac{1}{\mu_i}$ i.e. $\tilde{\mu}_i < \mu_i$. Let $\tilde{\lambda}_i > 0$ and $\lambda_i > 0$ be the loads allocated by the optimal algorithm when computer i bids $\tilde{b}_i$ and b_i, respectively. We must prove that $\tilde{\lambda}_i < \lambda_i$ i.e. the allocation function computed by the optimal algorithm is decreasing in b_i.

Assume by contradiction that $\tilde{\lambda}_i \geq \lambda_i$. This implies $1/(\mu_i - \lambda_i) < 1/(\tilde{\mu}_i - \lambda_i) \leq 1/(\tilde{\mu}_i - \tilde{\lambda}_i)$. This means that $\tilde{F}_i > F_i$. Since $\tilde{\lambda}_i > 0$ is higher than λ_i and $\sum_{i=1}^{n} \lambda_i = \Phi$ there must be a computer l such that $\tilde{\lambda}_l \leq \lambda_l$, $\lambda_l > 0$. Since $\tilde{\lambda}_i \geq \lambda_i > 0$ the Kuhn-Tucker conditions for optimality (5.13), (5.14) imply that:

$$\lambda_i F_i^{'} + F_i = \alpha \tag{C.1}$$

$$\tilde{\lambda}_i \tilde{F}_i^{'} + \tilde{F}_i = \tilde{\alpha} \tag{C.2}$$

Since $\lambda_l \geq \tilde{\lambda}_l$ and $\lambda_l > 0$ the Kuhn-Tucker conditions for optimality (5.13), (5.14) imply that:

$$\lambda_l F_l^{'} + F_l = \alpha \tag{C.3}$$

$$\tilde{\lambda}_l \tilde{F}_l^{'} + \tilde{F}_l \geq \tilde{\alpha} \tag{C.4}$$

Combining equations (C.1)-(C.4) we obtain:

$$\lambda_i F_i^{'} + F_i = \lambda_l F_l^{'} + F_l \tag{C.5}$$

$$\tilde{\lambda}_l \tilde{F}_l^{'} + \tilde{F}_l \geq \tilde{\lambda}_i \tilde{F}_i^{'} + \tilde{F}_i \tag{C.6}$$

Because $\tilde{\lambda}_l \leq \lambda_l$ we have $1/(\mu_l - \tilde{\lambda}_l) \leq 1/(\mu_l - \lambda_l)$. This implies $\tilde{F}_l \leq F_l$. Also, using $\tilde{\lambda}_l \leq \lambda_l$ we obtain the following equation:

$$\lambda_i F_i^{'} + F_i \geq \tilde{\lambda}_i \tilde{F}_i^{'} + \tilde{F}_i \tag{C.7}$$

This is a contradiction because $\tilde{\lambda}_i \geq \lambda_i$ and $\tilde{F}_i > F_i$. □

C.2 Proof of Theorem 5.2

We use the result of Archer and Tardos [7] that states that if the output function is decreasing in the bids then it admits a truthful payment scheme. We proved in Theorem 5.1 that the load function $\lambda(b)$ is decreasing in the bids, so it admits a truthful mechanism.

We next use another result from [7] stating that if the area under the work curve is finite the mechanism admits voluntary participation. For feasible bids, the area under the work curve is finite i.e.

$$\int_{b_{imin}}^{\infty} \lambda_i(b_{-i}, u) du < \infty$$

where b_{imin} is the bid that corresponds to $\lambda_i = \Phi$. Thus, our mechanism admits voluntary participation and the payments are given by equation (5.16). □

Appendix D

Proofs from Chapter 6

D.1 Proof of Theorem 6.1

The total latency function is a convex function and the optimal allocation can be obtained by using the Kuhn-Tucker conditions [94].

Let $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [94]. We consider the Lagrangian function:

$$\mathcal{L}(\mathbf{x}, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} a_i x_i^2 - \alpha(\sum_{i=1}^{n} x_i - R) - \sum_{i=1}^{n} \eta_i x_i \tag{D.1}$$

The Kuhn-Tucker conditions imply that x_i, $i = 1, \ldots, n$ is the optimal solution to our problem if and only if there exists $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ such that:

$$\frac{\partial \mathcal{L}}{\partial x_i} = 0, \qquad i = 1, \ldots, n \tag{D.2}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \tag{D.3}$$

$$\eta_i x_i = 0, \quad \eta_i \geq 0, \quad x_i \geq 0, \qquad i = 1, \ldots, n \tag{D.4}$$

We consider $x_i > 0$ and this implies that the last condition is satisfied only if $\eta_i = 0$. The conditions become:

$$2a_i x_i - \alpha = 0 \qquad i = 1, \ldots, n \tag{D.5}$$

$$\sum_{i=1}^{n} x_i - R = 0 \tag{D.6}$$

Solving these equations we get:

$$x_i = \frac{\alpha}{2a_i} \qquad i = 1, \ldots, n \tag{D.7}$$

$$\alpha = \frac{2R}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{D.8}$$

From these, we obtain the optimal allocation x_i:

$$x_i = \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}} R \qquad i = 1, \ldots, n \tag{D.9}$$

Using this allocation we compute the minimum value for the latency function as follows:

$$L^* = \sum_{i=1}^{n} a_i x_i^2 = \sum_{i=1}^{n} a_i \frac{(\frac{1}{a_i})^2}{(\sum_{k=1}^{n} \frac{1}{a_k})^2} R^2 = \frac{R^2}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{D.10}$$

□

D.2 Proof of Theorem 6.2

Assuming a vector of bids $\mathbf{b}$, the utility of agent i is:

$$U_i(\mathbf{b}, \tilde{\mathbf{t}}) = P_i(\mathbf{b}, \tilde{\mathbf{t}}) + V_i(\mathbf{x}(\mathbf{b}), \tilde{\mathbf{t}})$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i)) + \tilde{t}_i x_i^2(\mathbf{b}) - \tilde{t}_i x_i^2(\mathbf{b})$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i)) \tag{D.11}$$

We consider two possible situations:

i) $t_i = \tilde{t}_i$ i.e. agent i executes its assigned jobs using its full processing capability.

If agent i bids its true value t_i then its utility U_i^t is:

$$U_i^t = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, t_i), (\mathbf{b}_{-i}, \tilde{t}_i)) = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^t \tag{D.12}$$

If agent i bids lower ($b_i^l < t_i$) then its utility U_i^l is:

$$U_i^l = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, b_i^l), (\mathbf{b}_{-i}, \tilde{t}_i)) = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^l \tag{D.13}$$

We want to show that $U_i^t \geq U_i^l$, that reduces to show that $L_i^l \geq L_i^t$. Because L_i^t is the minimum possible value for the latency (from the optimality of PR algorithm), by bidding a lower value, agent i gets more jobs and the total latency is increased, thus $L_i^l \geq L_i^t$.

If agent i bids higher ($b_i^h > t_i$) then its utility U_i^h is:

$$U_i^h = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, b_i^h), (\mathbf{b}_{-i}, \tilde{t}_i)) = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^h \tag{D.14}$$

By bidding a higher value agent i gets fewer jobs and so more jobs will be assigned to the other agents. Due to the optimality of allocation the total latency increases i.e. $L_i^h \geq L_i^t$ and thus we have $U_i^t \geq U_i^h$.

ii) $\tilde{t}_i > t_i$ i.e. agent i executes its assigned jobs at a slower rate thus increasing the total latency. A similar argument as in the case i) applies. □

D.3 Proof of Theorem 6.3

The utility of agent i when it bids its true value t_i is:

$$U_i^t = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, t_i), (\mathbf{b}_{-i}, \tilde{t}_i)) \tag{D.15}$$

The latency L_{-i} is obtained by using all the other agents except agent i. By allocating the same number of jobs, we get a higher latency L_{-i} than in the case of using all the agents, with agent i bidding its true value (from the optimality of allocation). Thus $U_i^T \geq 0$. □

Bibliography

[1] *DIMACS Workshop on Computational Issues in Game Theory and Mechanism Design*, November 2001, DIMACS Center, Rutgers University. `http://dimacs.rutgers.edu/Workshops/gametheory/`.

[2] I. Ahmad and A. Ghafoor. Semi-distributed load balancing for massively parallel multicomputer systems. *IEEE Trans. Software Eng.*, 17(10):987–1003, October 1991.

[3] E. Altman. Flow control using the theory of zero sum Markov games. *IEEE Trans. Automatic Control*, 39(4):814–818, April 1994.

[4] E. Altman and T. Basar. Multi-user rate-based flow control: Distributed game-theoretic algorithms. In *Proc. of the 36th IEEE Conf. on Decision and Control*, pages 2916–2921, December 1997.

[5] E. Altman and T. Basar. Multiuser rate-based flow control. *IEEE Trans. Comm.*, 46(7):940–949, July 1998.

[6] E. Altman, T. Basar, T. Jimenez, and N. Shimkin. Routing in two parallel links: Game-theoretic distributed algorithms. *J. Parallel and Distributed Computing*, 61(9):1367–1381, September 2001.

[7] A. Archer and E. Tardos. Truthful mechanism for one-parameter agents. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science*, pages 482–491, October 2001.

[8] A. Archer and E. Tardos. Frugal path mechanisms. In *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 991–999, January 2002.

[9] A. Avritzer, M. Gerla, B. A. N. Ribeiro, J. W. Carlyle, and W. J. Karplus. The advantage of dynamic tuning in distributed asymmetric systems. In *INFOCOM '90, Proc. of the 9th Ann. Joint Conf. of the IEEE Computer and Communications Societies*, volume 3, pages 811–818, June 1990.

[10] S. A. Banawan and J. Zahorjan. Load sharing in heterogeneous queueing systems. In *INFOCOM '89, Proc. of the 8th Ann. Joint Conf. of the IEEE Computer and Communications Societies*, volume 2, pages 731–739, April 1989.

[11] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. SIAM, 1998.

[12] K. Benmohammed-Mahieddine, P. M. Dew, and M. Kara. A periodic symmetrically-initiated load balancing algorithm for distributed systems. In *Proc. of the 14th IEEE Intl. Conf. on Distributed Computing Systems*, pages 616–623, June 1994.

[13] F. Bonomi. On job assignement for parallel system of processor sharing queues. *IEEE Trans. Comput.*, 39(7):858–868, July 1990.

[14] F. Bonomi and A. Kumar. Adaptive optimal load balancing in heterogeneous multiserver system with a central job scheduler. In *Proc. of the 8th Intl. Conf. on Distributed Computing Systems*, pages 500–508, June 1988.

[15] F. Bonomi and A. Kumar. Optimality of weighted least squares load balancing. In *Proc. of the 27th IEEE Conf. on Decision and Control*, pages 1480–1485, December 1988.

[16] F. Bonomi and A. Kumar. Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler. *IEEE Trans. Comput.*, 39(10):1232–1250, October 1990.

[17] T. Boulogne, E. Altman, and O. Pourtallier. On the convergence to Nash equilibrium in problems of distributed computing. To appear *in Annals of Operation Research*, 2002.

[18] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service-oriented grid computing. In *Proc. of the 10th IEEE Heterogeneous Computing Workshop*, pages 776–790, April 2001.

[19] W. Cai, B. S. Lee, A. Heng, and L. Zhu. A simulation study of dynamic load balancing for network-based parallel processing. In *Proc. of the 3rd Intl. Symp. on Parallel Architectures, Algorithms and Networks*, pages 383–389, December 1997.

[20] L. M. Campos and I. Scherson. Rate of change load balancing in distributed and parallel systems. *Parallel Computing*, 26(9):1213–1230, July 2000.

[21] T. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Software Eng.*, 14(2):141–154, February 1988.

[22] H. W. D. Chang and W. J. B. Oldham. Dynamic task allocation models for large distributed computing systems. *IEEE Trans. Parallel and Distributed Syst.*, 6(12):1301–1315, December 1995.

[23] A. Chavez, A. Moukas, and P. Maes. *Challenger*: A multi-agent system for distributed resource allocation. In *Proc. of the Intl. Conf. on Autonomous Agents*, pages 323–331, February 1997.

[24] Y. C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.*, C-28(5):354–361, May 1979.

[25] E. Clarke. Multipart pricing of public goods. *Public Choice*, 8:17–33, 1971.

[26] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Trans. Networking*, 1(6):614–627, December 1993.

[27] A. Corradi, L. Leonardi, and F. Zambonelli. Diffusive load-balancing policies for dynamic applications. *IEEE Concurrency*, 7(1):22–31, Jan.-March 1999.

[28] A. Cortes, A. Ripoll, M. A. Senar, F. Cedo, and E. Luque. On the stability of a distributed dynamic load balancing algorithm. In *Proc. of the 1998 Intl. Conf. on Parallel and Distributed Systems*, pages 435–446, December 1998.

[29] A. Cortes, A. Ripoll, M. A. Senar, and E. Luque. Performance comparison of dynamic load balancing strategies for distributed computing. In *Proc. of the 32nd Hawaii Intl. Conf on System Sciences*, pages 170–177, February 1988.

[30] A. Cortes, A. Ripoll, M. A. Senar, P. Pons, and E. Luque. On the performance of nearest-neighbors load balancing algorithms in parallel systems. In *Proc. of the 7th Euromicro Workshop on Parallel and Distributed Processing*, page 10, January 1999.

[31] R. M. Cubert and P. Fishwick. *Sim++ Reference Manual*. CISE, Univ. of Florida, July 1995.

[32] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel and Distributed Computing*, 7(2):279–301, February 1989.

[33] S. P. Dandamudi. Sensitivity evaluation of dynamic load sharing in distributed systems. *IEEE Concurrency*, 6(3):62–72, July-Sept. 1998.

[34] E. de Souza e Silva and M. Gerla. Load balancing in distributed systems with multiple classes and site constraints. In *Performance '84*, pages 17–33, 1984.

[35] X. Deng and C. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, May 1994.

[36] S. Dierkes. Load balancing with a fuzzy-decision algorithm. *Information Sciences*, 97(1-2):159–177, March 1997.

[37] D. L. Eager, E. D. Lazowska, and J. Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. In *Proc. of the ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 1–3, 1985.

[38] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Software Eng.*, SE-12(5):662–675, May 1986.

[39] A. A. Economides. *A Unified Game-Theoretic Methodology for the Joint Load Sharing, Routing and Congestion Control Problem*. PhD thesis, Department of Computer Eng., University of Southern California, August 1990.

[40] A. A. Economides and J. Silvester. A game theory approach to cooperative and non-cooperative routing problems. In *ITS '90, Proc. of the Telecommunication Symp.*, pages 597–601, 1990.

[41] A. A. Economides and J. Silvester. Multi-objective routing in integrated services networks: A game theory approach. In *INFOCOM '91, Proc. of the 10th Ann. Joint Conf. of the IEEE Computer and Communications Societies*, volume 3, pages 1220–1227, April 1991.

[42] R. Elsasser, B. Monien, and R. Preis. Diffusive load balancing schemes on heterogeneous networks. In *Proc. of the 12th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pages 30–38, July 2000.

[43] S. C. Esquivel, G. M. Leguizamon, and R. H. Gallard. A hybrid strategy for load balancing in distributed systems environments. In *Proc. of the IEEE Intl. Conf. on Evolutionary Computation*, pages 127–132, April 1997.

[44] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. In *Proc. of the 21st ACM Symposium on Principles of Distributed Computing*, pages 173–182, July 2002.

[45] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proc. of the 32nd Annual ACM Symp. on Theory of Computing*, pages 218–227, May 2000.

[46] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. of the ACM Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, September 2002.

[47] D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In *Market based Control of Distributed Systems (ed. S. Clearwater, World Scientific Press*, 1996.

[48] D. Ferguson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed systems. In *Proc. of the 8-th IEEE Intl. Conf. on Distr. Comp. Systems*, pages 491–499, 1988.

[49] M. A. Franklin and V. Govindan. A general matrix iterative model for dynamic load balancing. *Parallel Computing*, 22(7):969–989, October 1996.

[50] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, 1994.

[51] B. Ghosh, S. Muthukrishnan, and M. H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *Proc. of the 8th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pages 72–81, June 1996.

[52] J. Gil. Renaming and dispersing: Techniques for fast load balancing. *J. Parallel and Distributed Computing*, 23(2):149–157, November 1994.

[53] K. K. Goswami, M. Devarakonda, and R. K. Iyer. Prediction-based dynamic load-sharing heuristics. *IEEE Trans. Parallel and Distributed Syst.*, 4(6):638–648, June 1993.

[54] A. R. Greenwald. *Learning to Play Network Games: Does Rationality Yield Nash Equilibrium?* PhD thesis, Department of Computer Science, New York University, May 1999.

[55] T. Groves. Incentive in teams. *Econometrica*, 41(4):617–631, 1973.

[56] C. C. Han, K. G. Shin, and S. K. Yun. On load balancing in multicomputer/distributed systems equipped with circuit or cut-through switching capability. *IEEE Trans. Comput.*, 49(9):947–957, September 2000.

[57] J. Harsanyi. Games with incomplete information played by Bayesian players, Part I. The basic model. *Management Science*, 14(3):159–182, November 1967.

[58] J. Harsanyi. Games with incomplete information played by Bayesian players, Part II. Bayesian equilibrium points. *Management Science*, 14(5):320–334, January 1968.

[59] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science*, pages 252–259, October 2001.

[60] M. T. T. Hsiao and A. A. Lazar. Optimal decentralized flow control of Markovian queueing networks with multiple controllers. *Performance Evaluation*, 13(3):181–204, 1991.

[61] Y. F. Hu and R. J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4):417–444, April 1999.

[62] B. A. Huberman and T. Hogg. Distributed computation as an economic system. *J. of Economic Perspectives*, 9(1):141–152, Winter 1995.

[63] C. C. Hui and S. T. Chanson. Improved strategies for dynamic load balancing. *IEEE Concurrency*, 7(3):58–67, July-Sept. 1999.

[64] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience, New York, NY, 1991.

[65] R. K. Jain, D.M. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report DEC-TR-301, Digital Equipment Corporation, Eastern Research Lab, 1984.

[66] H. Kameda, E. Altman, O. Pourtallier, J. Li, and Y. Hosokawa. Paradoxes in performance optimization of distributed systems. In *Proc. of SSGRR 2000 Computer and e-bussines Conf.*, July 2000.

[67] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer Verlag, London, 1997.

[68] H. Kameda and O. Pourtallier. A characterization of paradoxes in distributed optimization of performance for multiplicated systems. Technical Report ISE-TR-00-168, Inst. Information Sciences and Electronics, Univ. of Tsukuba, Japan, March 2000.

[69] H. Kameda and Y. Zhang. Uniqueness of the solution for optimal static routing in open BCMP queueing networks. *Mathl. Comput. Modelling*, 22(10-12):119–130, Nov.-Dec. 1995.

[70] R. Karp, E. Koutsoupias, C. H. Papadimitriou, and S. Shenker. Optimization problems in congestion control. In *Proc. of the 41st IEEE Symp. on Foundations of Computer Science*, pages 66–74, November 2000.

[71] C. Kim and H. Kameda. Optimal static load balancing of multi-class jobs in a distributed computer system. In *Proc. of the 10th Intl. Conf. on Distributed Computing Systems*, pages 562–569, May 1990.

[72] C. Kim and H. Kameda. An algorithm for optimal static load balancing in distributed computer systems. *IEEE Trans. Comput.*, 41(3):381–384, March 1992.

[73] L. Kleinrock. *Queueing Systems - Volume 1: Theory*. John Wiley and Sons, 1975.

[74] Y. A. Korilis and A. A. Lazar. On the existence of equilibria in noncooperative optimal flow control. *Journal of the ACM*, 42(3):584–613, May 1995.

[75] Y. A. Korilis, A. A. Lazar, and A. Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Trans. Networking*, 5(1):161–173, February 1997.

[76] Y. A. Korilis, A. A. Lazar, and A. Orda. Capacity allocation under noncooperative routing. *IEEE Trans. Automatic Control*, 42(3):309–325, March 1997.

[77] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Annual Symp. on Theoretical Aspects of Computer Science*, pages 404–413, 1999.

[78] O. Kremien and J. Kramer. Methodical analysis of adaptive load sharing algorithms. *IEEE Trans. Parallel and Distributed Syst.*, 3(6):747–760, November 1992.

[79] P. Krueger and N. G. Shivaratri. Adaptive location policies for global scheduling. *IEEE Trans. Software Eng.*, 20(6):432–444, June 1994.

[80] P. Kulkarni and I. Sengupta. A new approach for load balancing using differential load measurement. In *Proc. of Intl. Conf. on Information Technology: Coding and Computing*, pages 355–359, March 2000.

[81] V. Kumar, A. Y. Grama, and N. R. Vempaty. Scalable load balancing techniques for parallel computers. *J. Parallel and Distributed Computing*, 22(1):60–79, July 1994.

[82] R. La and V. Anantharam. Optimal routing control: Game theoretic approach. In *Proc. of the 36th IEEE Conf. on Decision and Control*, pages 2910–2915, December 1997.

[83] A. A. Lazar, A. Orda, and D. E. Pendarakis. Virtual path bandwidth allocation in multiuser networks. *IEEE/ACM Trans. Networking*, 5(6):861–871, December 1997.

[84] H. Lee. Optimal static distribution of prioritized customers to heterogeneous parallel servers. *Computers Ops. Res.*, 22(10):995–1003, December 1995.

[85] S. H. Lee and C. S. Hwang. A dynamic load balancing approach using genetic algorithm in distributed systems. In *Proc. of the IEEE Intl. Conf. on Evolutionary Computation*, pages 639–644, May 1998.

[86] S. H. Lee, T. W. Kang, M. S. Ko, G. S. Chung, J. M. Gil, and C. S. Hwang. A genetic algorithm method for sender-based dynamic load balancing algorithm in distributed systems. In *Proc. of the 1st Intl. Conf. on Knowledge-Based Intelligent Electronic Systems*, volume 1, pages 302–307, May 1997.

[87] W. Leinberger, G. Karypis, and V. Kumar. Load balancing across near-homogeneous multi-resource servers. In *Proc. of the 9th Heteorgeneous Computing Workshop*, pages 60–71, May 2000.

[88] J. Li and H. Kameda. A decomposition algorithm for optimal static load balancing in tree hierarchy network configuration. *IEEE Trans. Parallel and Distributed Syst.*, 5(5):540–548, May 1994.

[89] J. Li and H. Kameda. Optimal static load balancing in star network configurations with two-way traffic. *J. of Parallel and Distributed Computing*, 23(3):364–375, December 1994.

[90] J. Li and H. Kameda. Load balancing problems for multiclass jobs in distributed/parallel computer systems. *IEEE Trans. Comput.*, 47(3):322–332, March 1998.

[91] H. C. Lin and C. S. Raghavendra. A dynamic load-balancing policy with a central job dispatcher. *IEEE Trans. Software Eng.*, 18(2):148–158, February 1992.

[92] H. T. Liu and J. Silvester. An approximate performance model for load-dependent interactive queues with application to load balancing in distributed systems. In *INFOCOM ’88, Proc. of the 7th Ann. Joint Conf. of the IEEE Computer and Communications Societies*, pages 956–965, March 1988.

[93] R. D. Luce and H. Raiffa. *Games and Decisions*. Wiley, New York, 1957.

[94] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Mass., 1984.

[95] T. L. Magnanti. Models and algorithms for predicting urban traffic equilibria. In *Transportation Planning Models (ed. M. Florian)*, pages 153–185, 1984.

[96] M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proc. of the 33rd Annual ACM Symp. on Theory of Computing*, pages 510–519, July 2001.

[97] R. Mazumdar, L. G. Mason, and C. Douligeris. Fairness in network optimal flow control: Optimality of product forms. *IEEE Trans. Comm.*, 39(5):775–782, May 1991.

[98] R. Mirchandaney, D. Towsley, and J. Stankovic. Adaptive load sharing in heterogeneous systems. In *Proc. of the 9th IEEE Intl. Conf. on Distributed Computing Systems*, pages 298–306, June 1989.

[99] R. Mirchandaney, D. Towsley, and J. Stankovic. Analysis of the effects of delays on load sharing. *IEEE Trans. Comput.*, 38(11):1513–1525, November 1989.

[100] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proc. of the 9th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pages 292–301, June 1997.

[101] A. Muthoo. *Bargaining Theory with Applications*. Cambridge Univ. Press, Cambridge, U.K., 1999.

[102] J. Nash. The bargaining problem. *Econometrica*, 18(2):155–162, April 1950.

[103] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, September 1951.

[104] L. M. Ni and K. Hwang. Adaptive load balancing in a multiple processor system with many job classes. *IEEE Trans. Software Eng.*, SE-11(5):491–496, May 1985.

[105] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over Internet - The POPCORN project. In *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pages 592–601, May 1998.

[106] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. of the 31rd Annual ACM Symp. on Theory of Computing (STOC' 1999)*, pages 129–140, May 1999.

[107] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *Proc. of the 2nd ACM Conference on Electronic Commerce*, pages 242–252, October 2000.

[108] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35(1/2):166–196, April 2001.

[109] W. Obeloer, C. Grewe, and H. Pals. Load management with mobile agents. In *Proc. of the 24th Euromicro Conf.*, volume 2, pages 1005–1012, August 1998.

[110] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Networking*, 1(5):510–521, October 1993.

[111] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge, Mass., 1994.

[112] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. of the 41th IEEE Symp. on Foundations of Computer Science*, pages 86–92, November 2000.

[113] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of markov chains and the analysis of iterative load-balancing schemes. In *Proc. of the 39th Annual Symp. on Foundations of Computer Science*, pages 694–703, November 1998.

[114] R. Riedl and L. Richter. Classification of load distribution algorithms. In *Proc. of the 4th Euromicro Workshop on Parallel and Distributed Processing*, pages 404–413, Jan. 1996.

[115] J. B. Rosen. Existence and uniqueness of equilibrium points for n-person games. *Econometrica*, 33(3):520–534, July 1965.

[116] K. W. Ross and D. D. Yao. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM*, 38(3):676–690, July 1991.

[117] H. G. Rothitor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEE Proc.-Computers and Digital Techniques*, 141(1):1–10, Jan. 1994.

[118] T. Roughgarden. Stackelberg scheduling strategies. In *Proc. of the 33rd Annual ACM Symp. on Theory of Computing*, pages 104–113, July 2001.

[119] T. Roughgarden and E. Tardos. How bad is selfish routing? In *Proc. of the 41th IEEE Symp. on Foundations of Computer Science*, pages 93–102, November 2000.

[120] M. Schaar, K. Efe, and L. Delcambre L. N. Bhuyan. Load balancing with network cooperation. In *Proc. of the 11th IEEE Intl. Conf. on Distributed Computing Systems*, pages 328–335, May 1991.

[121] R. Schoonderwoerd, O. Holland, and J. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proc. of the 1st Intl. Conf. on Autonomous Agents*, pages 209–216, February 1997.

[122] S. Shenker and A. Weinrib. The optimal control of heterogeneous queueing systems: A paradigm for load-sharing and routing. *IEEE Trans. Comput.*, 38(12):1724–1735, December 1989.

[123] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *IEEE Computer Magazine*, 8(12):33–44, December 1992.

[124] J. A. Stankovic. An application of bayesian decision theory to decentralized control of job sceduling. *IEEE Trans. Comput.*, C-34(2):117–130, February 1985.

[125] A. Stefanescu and M. V. Stefanescu. The arbitrated solution for multi-objective convex programming. *Rev. Roum. Math. Pure Appl.*, 29:593–598, 1984.

[126] R. Subramanian and I. D. Scherson. An analysis of diffusive load-balancing. In *Proc. of the 6th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pages 220–225, June 1994.

[127] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *Proc. of the Intl. Conf. on Parallel Processing*, pages 373–382, August 2000.

[128] A. N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM*, 32(2):445–465, April 1985.

[129] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, March 1961.

[130] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Trans. Software Eng.*, 18(2):103–117, February 1992.

[131] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pages 612–621, May 1998.

[132] Y. T. Wang and R. J. T. Morris. Load sharing in distributed systems. *IEEE Trans. Comput.*, C-34(3):204–217, March 1985.

[133] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel and Distributed Syst.*, 4(9):979–993, September 1993.

[134] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-commerce: market formulations controlling resource allocation on the computational grid. In *Proc. of the 15th IEEE International Parallel and Distributed Processing Symposium*, April 2001.

[135] H. Yaiche, R. R. Mazumdar, and C. Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Trans. Networking*, 8(5):667–678, October 2000.

[136] K. M. Yu, S. J. Wu, and T. P Hong. A load balancing algorithm using prediction. In *Proc. of the 2nd Aizu Intl. Symp. on Parallel Algorithms/Architecture Synthesis*, pages 159–165, March 1997.

[137] Z. Zhang and C. Douligeris. Convergence of synchronous and asynchronous greedy algorithms in a multiclass telecommunications environment. *IEEE Trans. Comm.*, 40(8):1277–1281, August 1992.

Vita

Daniel Grosu was born in Constanţa, Romania on August 29, 1969, the son of Gherghina Grosu and Constantin Grosu. He received the Engineer Degree in Automatic Control and Industrial Informatics from The Technical University of Iasi, Romania in 1994 and his M.Sc. in Computer Science from The University of Texas at San Antonio in 2002. From 1994 to 1999 he was a lecturer in the Department of Electronics and Computers at Transylvania University, Brasov, Romania. He has held visiting research positions at the European Center for Parallelism of Barcelona and at the National Center for Microelectronics, Barcelona, Spain. His research interests include load balancing, distributed systems and topics at the border of computer science, game theory and economics. He has published 3 journal and 14 conference and workshop papers on these topics. He is a student member of the IEEE, ACM, SIAM and AMS.

Publications stemming from this dissertation are annotated here.

Publications

Journals

J1. D. Grosu and A.T. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems", *IEEE Transactions on Systems, Man and Cybernetics - Part B*, To appear. 2003.

J2. D. Grosu and A. T. Chronopoulos, "Noncooperative Load Balancing in Distributed Systems", *Journal of Parallel and Distributed Computing*, Submitted 2003.

J3. D. Grosu, A. T. Chronopoulos and M. Y. Leung, "Cooperative Load Balancing in Distributed Systems", *Journal of Parallel and Distributed Computing*, Submitted 2002.

Conferences

C1. D. Grosu and A.T. Chronopoulos, "A Truthful Mechanism for Fair Load Balancing in Distributed Systems", *Proc. of the 2nd IEEE International Symposium on Network Computing and Applications (NCA 2003)*, Cambridge, Massachusetts, USA, April 16-18, 2003 (accepted).

C2. D. Grosu and A. T. Chronopoulos, "A Load Balancing Mechanism with Verification", *Proc. of the 17th IEEE International Parallel and Distributed Processing Symposium, Workshop on Advances in Parallel and Distributed Computational Models (APDCM'03)*, Nice, France, April 22-26, 2003 (accepted).

C3. D. Grosu and A.T. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems", *Proc. of the 4th IEEE International Conference on Cluster Computing (CLUSTER 2002)*, pp. 445-450, Chicago, Illinois, USA, September 24-26, 2002.

C4. D. Grosu, A. T. Chronopoulos and M.Y. Leung, "Load Balancing in Distributed Systems: An Approach Using Cooperative Games", *Proc. of the 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002)*, pp. 501-510, Fort Lauderdale, Florida, USA, April 15-19, 2002.

C5. D. Grosu and A. T. Chronopoulos, "A Game-Theoretic Model and Algorithm for Load Balancing in Distributed Systems", *Proc. of the 16th IEEE International Parallel and Distributed Processing Symposium, Workshop on Advances in Parallel and Distributed Computational Models (APDCM'02)*, pp. 146-153, Fort Lauderdale, Florida, USA, April 15, 2002.

Other publications from previous research are annotated here.

Publications

Journals

J1. A. T. Chronopoulos, D. Grosu, A. M. Wissink, M. Benche and J. Liu, “An Efficient 3-D Grid Based Scheduling for Heterogeneous Systems”, *Journal of Parallel and Distributed Computing*, To appear. 2003.

J2. D. Grigoras and D. Grosu, “Simulation Results of an Associative Communication Protocol for Distributed Memory Parallel Systems”, *The Scientific Bulletin of The Polytechnic Institute of Iasi*, Romania, Vol. 38(42), No. 1-4, Section 4, pp. 33-40, 1992.

Conferences

C1. A. T. Chronopoulos, D. Grosu, A. M. Wissink and M. Benche, “Static Load Balancing for CFD Simulations on a Network of Workstations”, *Proc. of the IEEE International Symposium on Network Computing and Applications (NCA 2001)*, pp. 364-367, Cambridge, Massachusetts, USA, October 8-10, 2001.

C2. A.T. Chronopoulos, R. Andonie, M. Benche and D. Grosu, “A Class of Loop Self-Scheduling for Heterogeneous Clusters”, *Proc. of the 3rd IEEE International Conference on Cluster Computing (CLUSTER 2001)*, pp. 282-291, Newport Beach, California, USA, October 8-11, 2001.

C3. H. Galmeanu and D. Grosu, “A Parallel Block Matching Technique for MPEG-2 Motion Estimation”, *Proc. of the 3rd International Symposium on Video Processing and Multimedia Communications (VIPromCom 2001)*, pp. 121-124, Zadar, Croatia, June 13-15, 2001.

C4. R. Andonie, A.T. Chronopoulos, D. Grosu and H. Galmeanu, “Distributed Backpropagation Neural Networks on a PVM Heterogeneous System”, *Proc. of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS’98)*, Y. Pan, S.G. Akl, K. Li (eds.), pp. 555-560, Las Vegas, Nevada, USA, October 28-31, 1998.

This dissertation was typeset by Daniel Grosu in LaTeX.