

**GAME THEORY BASED JOB ALLOCATION/LOAD BALANCING IN
DISTRIBUTED SYSTEMS WITH APPLICATIONS TO GRID COMPUTING**

APPROVED BY SUPERVISING COMMITTEE:

Anthony T. Chronopoulos, Chair, Ph.D.

Clint Whaley, Ph.D.

Dakai Zhu, Ph.D.

Jeffery Von Ronne, Ph.D.

Yufei Huang, Ph.D.

ACCEPTED: _____
Dean, Graduate School

**GAME THEORY BASED JOB ALLOCATION/LOAD BALANCING IN
DISTRIBUTED SYSTEMS WITH APPLICATIONS TO GRID COMPUTING**

by

SATISH PENMATSA, 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

December 2007

ACKNOWLEDGEMENTS

I would like to express sincere gratitude for the time, encouragement, and guidance provided by my advisor, Dr. Anthony T. Chronopoulos. I also would like to thank Dr. Clint Whaley, Dr. Dakai Zhu, Dr. Jeffery Von Ronne, and Dr. Yufei Huang for serving on my dissertation committee.

I am grateful to my parents Sri Hari Raju and Vijaya Lakshmi, my wife Pratima, my in-laws Jagapathi Raju and Padmavathi, and all other family members 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.

This research was supported in part by the National Science Foundation under grant CCR-0312323. The research for this dissertation was conducted at the University of Texas at San Antonio within the laboratories of the Department of Computer Science.

December 2007

GAME THEORY BASED JOB ALLOCATION/LOAD BALANCING IN DISTRIBUTED SYSTEMS WITH APPLICATIONS TO GRID COMPUTING

Satish Penmatsa, Ph.D.
The University of Texas at San Antonio, 2007

Supervising Professor: Anthony T. Chronopoulos, Ph.D.

In this dissertation, we present job allocation or load balancing schemes for distributed systems and grid systems based on game theory. The main goal of our work is to provide fairness to the users and the users' jobs *i.e.*, all the users and their jobs should experience approximately equal expected response time (expected queuing delay + processing time + any communication time) or be charged approximately equal price for their execution independent of the computers allocated, and we will show that game theory provides a suitable framework for characterizing such schemes. Most of the previous work on load balancing did not take the fairness of allocation into account or considered fairness in a system without any communication costs.

For distributed systems in which all the jobs belong to a single user (single-class), we use a cooperative game to model the load balancing problem which takes the average system information into account (static load balancing). Our solution is based on the Nash Bargaining Solution which provides a Pareto optimal solution for the distributed system and is also a fair solution. We then extend the system model to include jobs from various users (multi-user/multi-class job distributed system) and include pricing to model a Grid system. For a grid system, we propose three static price-based job allocation schemes whose objective is to minimize the expected price for the grid users. One scheme provides a system optimal solution and is formulated as a constraint minimization problem and the other two schemes provide a fair solution and are formulated as non-cooperative games among the users. We use the concept of Nash equilibrium as the solution of our non-cooperative games and derive distributed algorithms for computing it. We also extend the proposed static load balancing schemes for multi-user jobs and formulate two schemes that take the current state of the system into account (dynamic load balancing). One dynamic scheme tries to minimize the expected response time of the entire system and the other tries to minimize the expected response time of the individual users to provide a fair solution.

TABLE OF CONTENTS

Acknowledgments	iii
Abstract	iv
List of Tables	ix
List of Figures	x
Chapter 1: Introduction	1
1.1 Job Allocation and Load Balancing in Distributed Systems	1
1.2 Related Work	2
1.3 Motivation for this Dissertation	4
1.4 Contributions of this Dissertation	4
1.5 Organization of the Dissertation	6
Chapter 2: Game Theory Concepts	7
2.1 Introduction	7
2.2 Examples from Economics	8
Chapter 3: Cooperative Load Balancing in Distributed Systems	13
3.1 Introduction	13
3.1.1 Related Work and Our Contribution	13
3.1.2 Chapter Organization	14
3.2 Cooperative Game Theory Concepts	14
3.3 Distributed System Model	16
3.4 Cooperative Load Balancing	19
3.5 Modeling Results	23
3.5.1 Effect of System Utilization	24
3.5.2 Effect of Heterogeneity	25

3.5.3	Effect of Communication Time	27
3.6	Summary	27

Chapter 4: Job Allocation Schemes for Computational Grids 29

4.1	Introduction	29
4.1.1	Related Work and Our Contribution	29
4.1.2	Chapter Organization	30
4.2	Pricing Model	30
4.3	Grid System Model	32
4.4	Price-based Job Allocation Schemes	35
4.4.1	Global Optimal Scheme with Pricing (GOSP)	35
4.4.2	Nash Scheme with Pricing (NASHP)	38
4.5	Modeling Results	39
4.5.1	Effect of System Utilization	40
4.5.2	Effect of Heterogeneity	42
4.6	Summary	44

Chapter 5: Job Allocation Scheme for Computational Grids with Communication

Costs		45
5.1	Introduction	45
5.1.1	Related Work and Our Contribution	45
5.1.2	Pricing Model	46
5.1.3	Chapter Organization	46
5.2	Grid System Model	46
5.3	Non-cooperative Game among the Users	49
5.4	Modeling Results	55
5.4.1	Effect of System Utilization	56
5.4.2	Effect of Heterogeneity	58
5.5	Summary	60

Chapter 6: Dynamic Load Balancing in Distributed Systems	61
6.1 Introduction	61
6.1.1 Related Work	61
6.1.2 Our Contribution	62
6.1.3 Chapter Organization	63
6.2 Distributed System Model	63
6.3 Static Load Balancing	64
6.3.1 Global Optimal Scheme (GOS)	66
6.3.2 Non-cooperative Scheme (NCOOPC)	67
6.4 Dynamic Load Balancing	71
6.4.1 Dynamic Global Optimal Scheme (DGOS)	71
6.4.2 Dynamic Non-cooperative Scheme (DNCOOPC)	75
6.5 Modeling Results	78
6.5.1 Effect of System Utilization	79
6.5.2 Effect of Bias (Δ)	82
6.5.3 Effect of Exchange Period (P)	82
6.6 Summary	83
 Chapter 7: Conclusions	 85
7.1 Summary and Contributions of this Dissertation	85
7.2 Future Research Directions	86
7.2.1 Load balancing in Distributed Systems and Grids with Mechanism Design	86
 Appendices	 88
Appendix A. Proofs for Chapter 3	88
A.1 Proof of Theorem 3.4.1	88
A.2 Proof of Theorem 3.4.2	88
A.3 Proof of Theorem 3.4.3	88
Appendix B. Proofs for Chapter 4	92
B.1 Proof of Theorem 4.4.1	92
B.2 Proof of Theorem 4.4.2	94

Appendix C. Proofs for Chapter 5	95
C.1 Proof of Theorem 5.3.1	95
Appendix D. Proofs for Chapter 6	98
D.1 Proof of Proposition 6.4.1	98
D.2 Proof of Proposition 6.4.2	99
Bibliography	100
Vita	

LIST OF TABLES

3.1	System configuration	24
3.2	Job arrival fractions q_i for each computer	24
3.3	System parameters	26
4.1	System configuration	40
5.1	System configuration	56
5.2	Job arrival fractions q^j for each user	57
5.3	System parameters	58
6.1	System configuration	78
6.2	Job arrival fractions q^j for each user	79

LIST OF FIGURES

2.1	Geometric representation of NBS	9
2.2	Firm 1's profit as a function of its output, given firm 2's output	11
2.3	The best response functions in Cournot's duopoly game. The unique Nash equilibrium is $(q_1^*, q_2^*) = (\frac{\alpha-c}{3}, \frac{\alpha-c}{3})$	12
3.1	Distributed System Model	16
3.2	System Utilization vs Expected Response Time	25
3.3	System Utilization vs Fairness Index	25
3.4	Heterogeneity vs Expected Response Time	26
3.5	Heterogeneity vs Fairness Index	27
3.6	Effect of Communication Time	28
3.7	Communication Time vs Expected Response Time	28
4.1	Bargaining game mapping between the grid servers and computers	31
4.2	Expected surplus of the Grid server vs Offered prices	32
4.3	Expected surplus of the Computer vs Offered prices	33
4.4	Grid System Model	33
4.5	System Utilization vs Expected Price (or Cost)	41
4.6	System Utilization vs Expected Response Time	42
4.7	System Utilization vs Fairness Index	42
4.8	Heterogeneity vs Expected Price	43
4.9	Heterogeneity vs Expected Response Time	43
5.1	Grid System Model	47
5.2	System Utilization vs Expected Response Time	57
5.3	System Utilization vs Fairness Index	58
5.4	Heterogeneity vs Expected Response Time	59
5.5	Heterogeneity vs Fairness Index	59
6.1	Distributed System Model	65
6.2	Variation of expected response time with system utilization ($OV = 0$)	80

6.3	Expected response time for each user ($OV = 0$)	81
6.4	Variation of expected response time with system utilization ($OV = 5\%$)	81
6.5	Variation of expected response time with system utilization ($OV = 10\%$)	82
6.6	Variation of expected response time of DNCOOPC with system utilization for various biases ($OV = 5\%$)	83
6.7	Variation of expected response time with exchange period ($\psi = 80\%$)	83

CHAPTER 1: Introduction

1.1 Job Allocation and Load Balancing in Distributed Systems

A distributed system often consists of heterogeneous computing and communication resources. Due to the possible differences in the computing capacities and uneven job arrival patterns, the workload on different computers in the system can vary greatly (Anderson, Culler, Patterson, and the NOW team 1995). This situation can lead to poor system performance. Improving the performance of such a system by an appropriate distribution of the workload among the computers is commonly known as *job allocation* or *load balancing*. Formally, this problem can be stated as follows: given a large number of jobs, find an allocation of jobs to the computers optimizing a given objective function (e.g., total expected (mean) response time (expected queuing delay + processing time + any communication time) of the system or total expected cost (the price that has to be paid by the users for using the resources) of the system).

There are two main categories of load balancing policies: *static policies* and *dynamic policies* (Shivaratri, Krueger, and Singhal 1992). Static policies base their decisions on collected statistical information about the system. They do not take into consideration the current state of the system. Dynamic policies base their decisions on the current state of the system, where state could refer to, for example, the number of jobs waiting in the queue to be processed and job arrival rate. The nodes (computers) exchange this information periodically and will try to balance the load by transferring some jobs from heavily loaded nodes to lightly loaded nodes. Despite the higher runtime complexity, dynamic policies can lead to better performance than static policies.

Jobs in a distributed system can be divided into different classes (multi-class or multi-user): (i) in multi-user type, each user jobs are grouped together; (ii) in multi-class type, all the jobs of the same size or of the same arrival rate are grouped together. So, the objective of the load balancing schemes can be to provide (1) a *system-optimal* solution where all the jobs are regarded to belong to one group (single-class) (2) an *individual-optimal* solution where each job optimizes its own expected response time (3) a *class-optimal (user-optimal)* solution where the jobs are classified into a finite number of classes (users) and each user tries to optimize the expected response time of his own jobs.

A *Grid* (Foster and Kesselman 2004) is a conglomeration of computing resources connected

by a network, which is used to solve large-scale computation problems. This system tries to run these applications by allocating the idle computing resources over a network or the internet commonly known as the *computational grid*. These computational resources have different owners who can be enabled by an automated negotiation mechanism by the grid controllers. The prices that the grid users have to pay for using the computing resources owned by different resource owners can be obtained using a pricing model based on a game theory framework. The objective of job allocation in grid systems can be to find an allocation that reduces the price that the grid users has to pay for utilizing the resources.

1.2 Related Work

Extensive studies exist on the static load balancing problem in single-class and multi-class job distributed systems. Most of those used the *global* approach, where the focus is on minimizing the expected response time of the entire system over all the jobs. Different network configurations are considered and the problem is formulated as a non-linear optimization problem in (Tantawi and Towsley 1985; Kim and Kameda 1992; Li and Kameda 1994; Tang and Chanson 2000; Kameda, Li, Kim, and Zhang 1997; Lee 1995) and as a polymatroid optimization problem in (Ross and Yao 1991). These schemes implement the entire system optimization approach in order to determine a load allocation that yields a system-wide optimal expected response time.

A few studies exist on static load balancing that provide individual-optimal and user-optimal solutions (Kameda, Li, Kim, and Zhang 1997) which are based on game theory. Individual and user-optimal policies for an infinite number of jobs/users based on *non-cooperative* games using *Wardrop equilibrium* are studied in (Kameda, Li, Kim, and Zhang 1997). An individual-optimal solution for finite jobs based on *cooperative* game theory is provided in (Grosu, Chronopoulos, and Leung 2002). User-optimal solutions based on *Nash equilibrium* are provided in ((Grosu and Chronopoulos 2005) and references therein) for finite number of users. Game theory is also used to model grid systems ((Kwok, Hwang, and Song 2007; Ghosh, Roy, Das, and Basu 2005; Kwok, Song, and Hwang 2005; Grosu and Das 2004) and references therein) and for price-based job allocation in distributed systems ((Ghosh, Basu, and Das 2007; Ghosh, Roy, Basu, and Das 2004; Ghosh, Basu, and Das 2005) and references therein).

Also, dynamic load balancing policies exist which can be distinguished as: *centralized* and *distributed* ((El-Zoghdy, Kameda, and Li 2002; Anand, Ghose, and Mani 1999; Zeng and Veeravalli 2006; Akay and Erciyes 2003; Choi 2004; Zeng and Bharadwaj 2004) and references therein). (a) *Centralized policies*: In this type of policies a dedicated computer is responsible for maintaining a global state of the system ((Hui and Chanson 1999) and references therein). Based on the collected information, the central computer makes allocation decisions. Many centralized schemes were proposed based on queuing models ((Cai, Lee, Heng, and Zhu 1997; Mitzenmacher 1997) and references therein) and non-queuing models ((Kulkarni and Sengupta 2000; Han, Shin, and Yun 2000; Hui and Chanson 1999) and references therein). For a large number of computers, the overhead of such schemes become prohibitive and the central decision computer becomes a bottleneck. (b) *Distributed policies*: In this type of policies, each computer constructs its own view of the global state ((Campos and Scherson 2000) and references therein). Most of the distributed schemes are sub-optimal and heuristic due to unavailability of accurate and timely global information. Distributed dynamic load balancing schemes can be classified as: *sender-initiated* ((Shivaratri, Krueger, and Singhal 1992; Dandamudi 1998) and references therein), *receiver-initiated* ((Shivaratri, Krueger, and Singhal 1992) and references therein), and *symmetrically-initiated* ((Shivaratri, Krueger, and Singhal 1992; Benmohammed-Mahieddine, Dew, and Kara 1994) and references therein).

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 diffusion schemes and schemes that use concepts from artificial intelligence (Corradi, Leonardi, and Zambonelli 1999; El-sasser, Monien, and Preis 2000; Cortes, Ripoll, Senar, and Luque 1999; Cortes, Ripoll, Senar, Pons, and Luque 1999; Ghosh, Muthukrishnan, and Schultz 1996; Rabani, Sinclair, and Wanka 1998). Load balancing based on genetic algorithms was studied in (Esquivel, Leguizamon, and Gallard 1997; Lee and Hwang 1998; Lee, Kang, Ko, Chung, Gil, and Hwang 1997). There are no known pure game theoretic studies on dynamic load balancing, but there exists some related approaches derived from economic theory (Ferguson, Nikolaou, Sairamesh, and Yemini 1996; Chavez, Moukas, and Maes 1997).

1.3 Motivation for this Dissertation

Most of the previous work on static and dynamic load balancing in distributed systems considered optimization of the entire system and considered expected response time as their main objective function. ((Kameda, Li, Kim, and Zhang 1997; Li and Kameda 1994; Tantawi and Towsley 1985; Anand, Ghose, and Mani 1999; Zeng and Veeravalli 2006; Choi 2004) and references therein). So, individual users' jobs may get delayed *i.e.*, some users may experience longer expected response times than others. This may not be acceptable in current distributed systems, where users have requirements for fast job execution. Studies for load balancing in distributed systems based on game theory that provides fairness (Jain 1991) to the users and the users' jobs (*i.e.*, all the users and their jobs experience approximately equal expected response time) did not take the communication subsystem into account. Also, game theory related work on job allocation in grid systems did not take the communication-subsystem or fairness-to-the-users into account.

1.4 Contributions of this Dissertation

In this study, we propose job allocation/load balancing schemes for distributed systems and grid systems based on game theory. The main goal of our work is to provide fairness to the users and the users' jobs *i.e.*, all the users and their jobs should experience approximately equal expected response time or be charged approximately equal expected price for their execution independent of the computers allocated. Our schemes will be suitable for systems in which the fair treatment of the users' jobs is as important as other performance characteristics.

We consider a distributed computer system that consists of heterogeneous host computers (nodes) interconnected by a communication network. We assume that "jobs of equal size and belonging to the same class" arrive at each computer with different arrival rates following an exponential (probability) distribution (Jain 1991; Grosu, Chronopoulos, and Leung 2002). Each computer determines whether a job should be processed locally or scheduled on a different computer for remote processing. Load balancing is achieved by transferring some jobs from nodes that are heavily loaded to the nodes that are idle or lightly loaded. A communication delay will be incurred as a result of sending a job to a different computer for processing. The communication delay is included in the model. We assume that the communication delay between two nodes is independent of the two nodes

but depends on the total traffic in the network. Examples of such a case are (i) local area networks, where all the nodes are competing for the same communication channel and so the communication delay between any two nodes depends on the total traffic generated by all the nodes; and (ii) satellite communication systems, where all the stations share the same communication media, and, therefore, the communication delay between two stations depends on the total traffic (Tantawi and Towsley 1985). This load balancing problem is formulated as a cooperative game among the computers and the communication subsystem. Based on the *Nash Bargaining Solution* (NBS) (Muthoo 1999; Nash 1950; Stefanescu and Stefanescu 1984) which provides a Pareto optimal (Mas-Collel, Whinston, and Green 1995) and fair solution, we provide a static algorithm for computing the NBS for our cooperative load balancing game. The main goal of this static load balancing scheme is to provide fairness to all the jobs, *i.e.*, all the jobs should experience the same expected response time independent of the allocated computer.

The single-class job distributed system model is then extended to a multi-class job distributed system model. Then pricing is included (the amount that has to be paid for using the resources) to model a grid system. Without taking the communication costs into account, we propose two static price-based job allocation schemes whose objective is to minimize the expected price of the grid users. The two schemes differ in their objective. One tries to minimize the expected price of the entire grid community (*i.e.*, all the grid users) to provide a system-optimal solution whereas the other tries to minimize the expected price of the individual grid users to provide a fair solution (*i.e.*, all the users should pay the same expected price for executing their jobs independent of the allocated computer). The prices that the grid users has to pay are obtained based on an incomplete information alternating-offer non-cooperative bargaining game between the grid servers (playing on behalf of the grid users) and the resource owners (computers).

Considering the communication costs in the above grid system model, we propose another static price-based job allocation scheme whose objective is to provide fairness to the grid users. This scheme is formulated as a non-cooperative game among the grid users who try to minimize the expected price of their own jobs.

The above static price-based job allocation schemes for multi-class job grid systems are then extended to dynamic load balancing schemes for multi-class (multi-user) job distributed systems. Taking the communication costs into account, we propose two dynamic load balancing schemes.

The first dynamic scheme tries to dynamically balance the load among the computers to obtain a system-optimal solution, which may not be fair to the users. The second dynamic scheme tries to dynamically balance the load among the computers to obtain a user-optimal solution. This solution provides fairness to all the users so that all the users have approximately the same expected response time independent of the computers allocated for their jobs.

We run a computer model with various system loads and configurations to evaluate the performance of our proposed load balancing/job allocation schemes.

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we present a brief review of some game theory concepts. In Chapter 3, we present a static cooperative load balancing scheme for single-class job distributed systems. In Chapter 4, we present two price-based job allocation schemes for computational grids. Chapter 5 presents another price-based job allocation scheme for grid systems by taking the communication subsystem into account. In Chapter 6, we propose two dynamic load balancing schemes for multi-class job distributed systems. In Chapter 7, we make conclusions and present possible directions for future work.

CHAPTER 2: Game Theory Concepts

In this chapter, we briefly review some important concepts from game theory (Fudenberg and Tirole 1994).

2.1 Introduction

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

There are two main classes of games: *cooperative games* in which the players have complete freedom of pre-play communications to make joint agreements and *non-cooperative games* in which no pre-play communication is permitted between the players (Luce and Raiffa 1957). 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* (Fudenberg and Tirole 1994).

One of the cornerstones of non-cooperative game theory is the notion of *Nash equilibrium* (Nash 1950; Nash 1951). 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 minimize his objective function, then, the Nash equilibrium can be defined as follows:

A strategy profile s^* is a Nash equilibrium if for all players $i \in \mathcal{I}$:

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

In other words, no player can decrease the value of his 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:

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

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* (Harsanyi 1967; Harsanyi 1968). This type of game is also called a *Bayesian game*. If the objective functions are common knowledge, the game is said to have *complete information*.

For more details on game theory, we refer to (Basar and Olsder 1998; Fudenberg and Tirole 1994). In the following we present examples to illustrate some of the concepts presented above.

2.2 Examples from Economics

Example 2.2.1. Cooperative game

Assume two firms (players), A and B, can have a best profit of $x = a = 2$ units and $y = b = 3$ units respectively without any cooperation. If they cooperate, they can share a total profit of 10 units (*i.e.*, $x + y = 10$). Assume the profit function for A is $f_1(x, y) = (x - 2)^{1/2}$ and for B is $f_2(x, y) = (y - 3)^{1/2}$. If their bargaining strengths are equal (say, $h = k = 1/2$, $h + k = 1$), what is the cooperative solution?

Solution: This game was proved to be equivalent to the Nash Bargaining Solution (NBS) problem (Dixit and Skeath 2004):

$$\text{maximize}_{x,y} (x - 2)^{1/2} (y - 3)^{1/2}$$

subject to the constraint $x + y = 10$. The solution is, $x = 4.5$ and $y = 5.5$.

Remark 2.2.1. In general, the problem is to choose x and y to maximize $(x - a)^h (y - b)^k$ subject to the constraint $y = g(x)$. For linear $g(x)$, the solution Q is the intersection of lines $x + y = v$ and $(y - b) = \frac{k}{h}(x - a)$ (Figure 2.1).

Example 2.2.2. Non-cooperative game (Cournot's Duopoly)

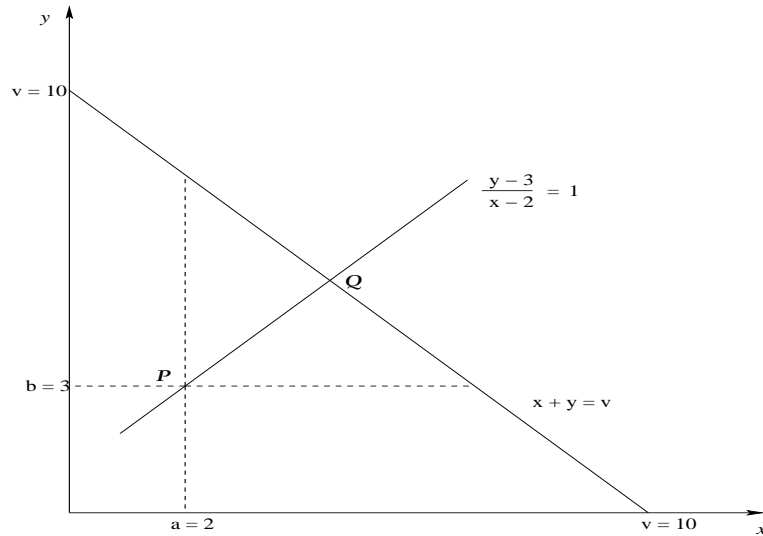


Figure 2.1: Geometric representation of NBS

Cournot competition (Osborne 2004) is an economics model used to describe industry structure. A single good is produced by n firms. The cost to firm i of producing q_i units of good is $C_i(q_i)$, where C_i is an increasing function (more output is more costly to produce). All the output is sold at a single price, determined by the demand for the good and the firms' total output. Specifically, if the firms' total output is Q , then the market price is $P(Q)$; P is called the *inverse demand function* (because, as the output increases, the market price decreases). If the output of each firm i is q_i , then the price is $P(q_1 + \dots + q_n)$, so that firm i 's revenue is $q_i P(q_1 + \dots + q_n)$. Thus firm i 's profit (π_i), equal to its revenue minus its cost, is

$$\pi_i(q_1, \dots, q_n) = q_i P(q_1 + \dots + q_n) - C_i(q_i) \quad (2.3)$$

One model of duopoly (only two firms) is the game in which

- the players are the firms.
- the actions of each firm are the set of possible outputs (any nonnegative amount).
- the payoff of each firm is its profit.

Suppose there are two firms and each firm's cost function is given by $C_i(q_i) = cq_i$ for all q_i (unit cost

is constant, equal to c), and the inverse demand function is linear where it is positive, given by

$$P(Q) = \begin{cases} \alpha - Q & \text{if } Q \leq \alpha \\ 0 & \text{if } Q > \alpha. \end{cases} \quad (2.4)$$

where $\alpha > 0$ and $c \geq 0$ are constants. If the firms' outputs are q_1 and q_2 , then the market price $P(q_1 + q_2) = \alpha - q_1 - q_2$ if $q_1 + q_2 < \alpha$ and zero if $q_1 + q_2 > \alpha$. Thus firm 1's profit is

$$\pi_1(q_1, q_2) = q_1(P(q_1 + q_2) - c) = \begin{cases} q_1(\alpha - c - q_1 - q_2) & \text{if } q_1 + q_2 \leq \alpha \\ -cq_1 & \text{if } q_1 + q_2 > \alpha \end{cases} \quad (2.5)$$

To find the Nash equilibria in this example, we can use the procedure based on the firms' best response functions. To find firm 1's best response to any given output q_2 of firm 2, we need to study firm 1's profit as a function of its output q_1 for given values of q_2 . Firm 1's best response function gives, for each possible output of firm 2, the profit-maximizing output of firm 1. If $q_2 = 0$ then firm 1's profit is $\pi_1(q_1, 0) = q_1(\alpha - c - q_1)$ for $q_1 \leq \alpha$, a quadratic function that is zero when $q_1 = 0$ and when $q_1 = \alpha - c$. Given the symmetry of quadratic functions, the output q_1 of firm 1 that maximizes its profit is $q_1 = \frac{1}{2}(\alpha - c)$. This can be obtained by setting the derivative of firm 1's profit with respect to q_1 equal to zero and solving for q_1 . Thus firm 1's best response to an output of zero for firm 2 is $b_1(0) = \frac{1}{2}(\alpha - c)$.

As the output q_2 of firm 2 increases, the profit firm 1 can obtain at any given output decreases, because more output of firm 2 means a lower price. This can be seen in Figure 2.2 (inner black curve) for $\pi_1(q_1, q_2)$ for $q_2 > 0$ and $q_2 < \alpha - c$. In this case, we have the output that maximizes $\pi_1(q_1, q_2) = q_1(\alpha - c - q_2 - q_1)$ as $q_1 = \frac{1}{2}(\alpha - c - q_2)$ for $q_2 > 0$ and $q_2 < \alpha - c$. Therefore, we conclude that the best response of firm 1 is given by

$$b_1(q_2) = \begin{cases} \frac{1}{2}(\alpha - c - q_2) & \text{if } q_2 \leq \alpha - c \\ 0 & \text{if } q_2 > \alpha - c. \end{cases} \quad (2.6)$$

Because firm 2's cost function is the same as firm 1's, its best response function $b_2(q_1)$ is also the same: for any number q , we have $b_2(q) = b_1(q)$. Of course, firm 2's best response function associates a value of firm 2's output with every output of firm 1, whereas firm 1's best response

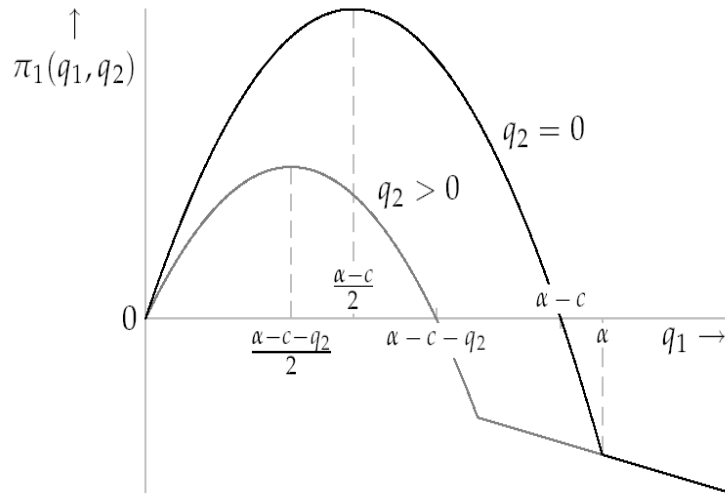


Figure 2.2: Firm 1's profit as a function of its output, given firm 2's output

function associates a value of firm 1's output with every output of firm 2, so we plot them relative to different axes. They are shown in Figure 2.3 (b_1 is dark; b_2 is light). In general, b_1 associates each point on the vertical axis with a point on the horizontal axis, and b_2 associates each point on the horizontal axis with a point on the vertical axis.

A Nash equilibrium is a pair (q_1^*, q_2^*) of outputs for which q_1 is a best response to q_2 , and q_2 is a best response to q_1 :

$$q_1^* = b_1(q_2^*) \quad \text{and} \quad q_2^* = b_2(q_1^*) \quad (2.7)$$

The set of such pairs is the set of points at which the best response functions in Figure 2.3 intersect. From the figure we see that there is exactly one such point, which is given by the solution of the two equations

$$\begin{aligned} q_1 &= \frac{1}{2}(\alpha - c - q_2) \\ q_2 &= \frac{1}{2}(\alpha - c - q_1) \end{aligned} \quad (2.8)$$

Solving these two equations, we have $q_1 = q_2 = \frac{\alpha - c}{3}$.

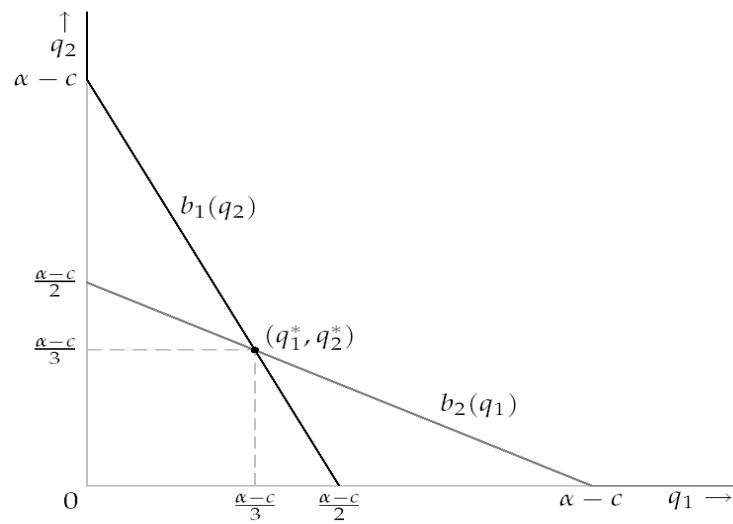


Figure 2.3: The best response functions in Cournot's duopoly game. The unique Nash equilibrium is $(q_1^*, q_2^*) = (\frac{\alpha - c}{3}, \frac{\alpha - c}{3})$

CHAPTER 3: Cooperative Load Balancing in Distributed Systems

3.1 Introduction

In this chapter (Penmatsa and Chronopoulos 2006a), we consider the static load balancing problem for single-class jobs in a distributed computer system that consists of heterogeneous host computers (nodes) interconnected by a communication network. Load balancing is achieved by transferring some jobs from nodes that are heavily loaded to those that are idle or lightly loaded. A communication delay will be incurred as a result of sending a job to a different computer for processing.

The load balancing problem is formulated as a cooperative game among the computers and the communication subsystem and game theory offers a suitable modeling framework (Fudenberg and Tirole 1994). The several decision makers (e.g., computers and the communication subsystem) cooperate in making decisions such that each of them will operate at its optimum. Based on the *Nash Bargaining Solution* (NBS) which provides a Pareto optimal and fair solution, we provide an algorithm for computing the NBS for our cooperative load balancing game.

3.1.1 Related Work and Our Contribution

The static load balancing problem in distributed systems has been extensively studied ((Tantawi and Towsley 1985; Li and Kameda 1994; Tang and Chanson 2000; Kameda, Li, Kim, and Zhang 1997) and references therein). Different network configurations were considered and the problem was formulated as a non-linear optimization problem. All these schemes determine a load allocation in order to obtain a system-optimal solution. Some game theory based load balancing schemes that provide individual-optimal and user-optimal solutions were studied in ((Grosu, Chronopoulos, and Leung 2002; Grosu and Chronopoulos 2005; Kameda, Li, Kim, and Zhang 1997; Roughgarden 2001) and references therein).

The past works on individual-optimal or user-optimal schemes do not take the communication subsystem into account. In our study, we consider a distributed system which includes communication delays. Our main goal here is to obtain an individual-optimal scheme which provides fairness to all (user) jobs. This means that all the jobs should experience the same expected response time independent of the computer to which they are allocated. The fairness of allocation is an im-

portant factor in modern distributed systems and our scheme will be suitable for systems in which the fair treatment of the users' jobs is as important as other performance characteristics. We show that our cooperative load balancing scheme provides both fairness and a Pareto optimal operating point for the entire system. We run a computer model with various system loads and configurations to evaluate the performance of our cooperative load balancing scheme.

3.1.2 Chapter Organization

The rest of the chapter is organized as follows. In Section 3.2, we present some related cooperative game theory concepts. In Section 3.3, we present the distributed system model considered. The load balancing problem is formulated as a cooperative game in Section 3.4. The performance of the proposed scheme is evaluated in Section 3.5. A summary of the chapter is provided in Section 3.6.

3.2 Cooperative Game Theory Concepts

In this section, we summarize some concepts and results from cooperative game theory which are used later in the chapter.

Definition 3.2.1. A cooperative game

A cooperative game consists of:

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

We are interested in finding solutions for the cooperative game defined above that are Pareto optimal.

Definition 3.2.2. Pareto optimality (*Mas-Collel, Whinston, and Green 1995*)

The point $u \in U$ is said to be Pareto optimal if for each $v \in U$, $v \leq u$, then $v = u$. Here $U, U \subset \mathbf{R}^N$ is the set of achievable performances (Yaiche, Mazumdar, and Rosenberg 2000).

Definition 3.2.3. The Nash Bargaining Solution (NBS) (*Muthoo 1999; Nash 1950; Stefanescu and Stefanescu 1984*)

A mapping $S : G \rightarrow \mathbf{R}^N$ is said to be a NBS if:

- i) $S(U, \mathbf{u}^0) \in U_0$;
- ii) $S(U, \mathbf{u}^0)$ is Pareto optimal;

and satisfies the fairness axioms (Nash 1950). Here G denotes the set of achievable performances with respect to the initial agreement point (Yaiche, Mazumdar, and Rosenberg 2000).

Definition 3.2.4. Bargaining point (*Stefanescu and Stefanescu 1984*)

\mathbf{u}^* is a (*Nash*) bargaining point if it is given by $S(U, \mathbf{u}^0)$ and $\mathbf{f}^{-1}(\mathbf{u}^*)$ is called the set of (*Nash*) bargaining solutions.

The following characterization of the Nash bargaining point forms the basis for the results later in the chapter.

Theorem 3.2.1. Nash bargaining point characterization (*Stefanescu and Stefanescu 1984; Yaiche, Mazumdar, and Rosenberg 2000*)

Consider the assumptions from the above definitions and references therein. Let J denote the set of players who are able to achieve a performance strictly superior to their initial performance and let X_0 denote the set of strategies that enable the players to achieve at least their initial performances. Let the vector function $\{f_j\}, j \in J$ be one-to-one on X_0 . Then, there exists a unique bargaining point \mathbf{u}^* and the set of the bargaining solutions $\mathbf{f}^{-1}(\mathbf{u}^*)$ is determined by solving the following optimization problems:

$$(P_J) : \quad \min_{\mathbf{x}} \prod_{j \in J} (f_j(\mathbf{x}) - u_j^0) \quad \mathbf{x} \in X_0 \quad (3.9)$$

$$(P'_J) : \quad \min_{\mathbf{x}} \sum_{j \in J} \ln(f_j(\mathbf{x}) - u_j^0) \quad \mathbf{x} \in X_0 \quad (3.10)$$

Then, (P_J) and (P'_J) are equivalent. (P'_J) is a convex optimization problem and has a unique solution. The unique solution of (P'_J) is the bargaining solution. \square

3.3 Distributed System Model

We consider a distributed system model with n nodes (or computers) connected by a communication network as shown in Figure 3.1.

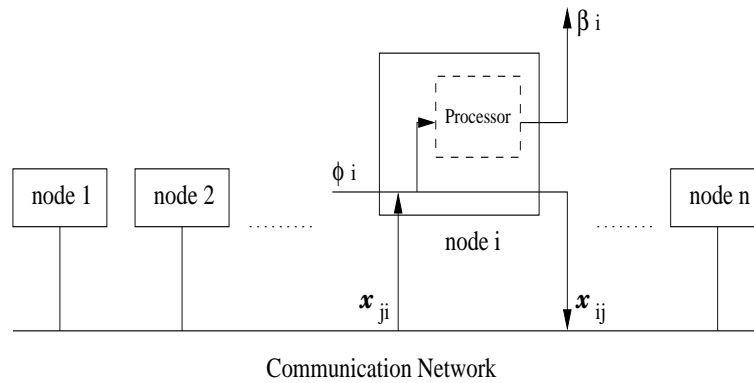


Figure 3.1: Distributed System Model

The terminology and assumptions used are as follows:

- The computers and the communication network are assumed to be product-form queuing network models (Jain 1991). Jobs arrive in a single queue at each node (computer) which has only one computing resource.
- ϕ_i denotes the external job arrival rate at node i . All the jobs in the system are assumed to be of the same size.
- The total external job arrival rate of the system is denoted by Φ . So, $\Phi = \sum_{i=1}^n \phi_i$.
- The job processing rate (load) at node i is denoted by β_i .
- x_{ij} denotes the job flow rate from node i to node j .
- A job arriving at node i may be either processed at node i or transferred to another node j through the communication network. The decision of transferring a job does not depend on the state of the system and hence is *static* in nature.

- A job transferred from node i to node j receives its service at node j and is not transferred to other nodes. If a node i sends (receives) jobs to (from) node j , node j does not send (receive) jobs to (from) node i .

The response time of a job in a system as above consists of a node delay (queuing delay + processing delay) at the processing node and also some possible communication delay incurred due to a job transfer. We model each node and the communication network as M/M/1 queuing systems (Jain 1991; Kleinrock 1975; Tantawi and Towsley 1985). In these queuing systems, the inter-arrival times and the service times are exponentially distributed (Jain 1991).

Let μ_i denote the processing (service) rate of node i , D_i denote the mean node delay or the expected response time of a job at node i , and E_i denote the mean number of jobs at node i . The mean node delay at a node can be calculated using Little's law (Jain 1991), which states that:

$$E_i = \phi_i \times D_i \quad (3.11)$$

Let ρ_i denote the *traffic intensity at node i* where $\rho_i = \frac{\phi_i}{\mu_i}$. Thus, we have (Jain 1991)

$$E_i = \frac{\rho_i}{(1 - \rho_i)} \quad (3.12)$$

Therefore, from eqs. (3.11) and (3.12), we have

$$D_i = \frac{E_i}{\phi_i} = \frac{\rho_i}{(1 - \rho_i)} \times \frac{1}{\phi_i} = \frac{1}{(\mu_i - \phi_i)} \quad (3.13)$$

Thus, the expected response time of a job at node i is given (as a function of ϕ_i) by:

$$D_i(\phi_i) = \frac{1}{\mu_i - \phi_i} \quad (3.14)$$

Since, our goal is to find the optimal ϕ_i (*i.e.*, β_i) at each node, we rewrite the above equation in terms of β_i . Thus, the mean node delay of a job at node i is given by:

$$D_i(\beta_i) = \frac{1}{\mu_i - \beta_i}, \quad i = 1, \dots, n. \quad (3.15)$$

We assume that the expected communication delay from node i to node j is independent of the source-destination pair (i, j) but may depend on the total traffic through the network denoted by λ where $\lambda = \sum_{i=1}^n \sum_{j=1}^n x_{ij}$. We denote the mean communication delay for a job by $G(\lambda)$. Modeling the communication network as an M/M/1 queuing system gives:

$$G(\lambda) = \frac{t}{1 - t\lambda}, \quad \lambda < \frac{1}{t} \quad (3.16)$$

where t is the mean communication time for sending and receiving a job. Note that $D_i(\beta_i)$ and $G(\lambda)$ are increasing positive functions. We classify the nodes in the following way as in (Tantawi and Towsley 1985):

- Sink (S): only receives jobs from other nodes but it does not send out any jobs.
- Idle source (R_d): does not process any jobs ($\beta_i = 0$) and it sends all the jobs to other nodes. It does not receive any jobs from other nodes.
- Active source (R_a): processes some of the jobs that arrive and it sends the remaining jobs to other nodes. But, it does not receive any jobs from other nodes.
- Neutral (N): processes jobs locally without sending or receiving jobs.

The network traffic λ can be expressed in terms of the variable β_i as

$$\lambda = \frac{1}{2} \sum_{i=1}^n |\phi_i - \beta_i| \quad (3.17)$$

We define the following functions:

$$d_i(\beta_i) = \frac{\partial}{\partial \beta_i} \ln D_i(\beta_i) = \frac{1}{\mu_i - \beta_i} \quad (3.18)$$

$$g(\lambda) = \frac{\partial}{\partial \lambda} \ln G(\lambda) = \frac{t}{(1 - t\lambda)} \quad (3.19)$$

$$d_i^{-1}(x) = \begin{cases} \mu_i - \frac{1}{x}, & \text{if } x > \frac{1}{\mu_i} \\ 0, & \text{if } x \leq \frac{1}{\mu_i} \end{cases} \quad (3.20)$$

3.4 Cooperative Load Balancing

In this section, we formulate the load balancing problem as a cooperative game among the computers and the communication network. We consider an $n + 1$ player game where the n computers try to minimize their expected response time $D_i(\beta_i)$ and the $(n + 1)$ th player, the communication subsystem, tries to minimize the expected communication delay $G(\lambda)$. So, the objective function for each computer i , $i = 1, \dots, n$ can be expressed as:

$$f_i(X) = D_i(\beta_i) \quad (3.21)$$

and the objective function for the communication subsystem can be expressed as:

$$f_{n+1}(X) = G(\lambda) \quad (3.22)$$

where $X = [\beta_1, \dots, \beta_n, \lambda]^T$ is the set of strategies of the $n + 1$ players.

Definition 3.4.1. The cooperative load balancing game

The cooperative load balancing game consists of:

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

$$\beta_i < \mu_i, \quad i = 1, \dots, n \quad (3.23)$$

$$\sum_{i=1}^n \beta_i = \sum_{i=1}^n \phi_i = \Phi, \quad (3.24)$$

$$\beta_i \geq 0, \quad i = 1, \dots, n \quad (3.25)$$

- For each computer i , $i = 1, \dots, n$, the *objective function* $f_i(X) = D_i(\beta_i)$; for the communication subsystem, the *objective function* $f_{n+1}(X) = G(\lambda)$; $X = [\beta_1, \dots, \beta_n, \lambda]^T$. The goal is to minimize simultaneously all $f_i(X)$, $i = 1, \dots, n + 1$.
- For each player i , $i = 1, \dots, n + 1$, the initial performance $u_i^0 = f_i(X^0)$, where X^0 is a zero vector of length $n + 1$.

Remark 3.4.1. In the above definition, we can assume that $\beta_i \leq \hat{\mu}_i$ to satisfy the compactness requirement for X , where $\hat{\mu}_i = \mu_i - \epsilon$ for a small $\epsilon > 0$. We ignore this condition for simplicity. We also assume that all the players in the above game are able to achieve performance strictly superior to their initial performance.

Theorem 3.4.1. For the cooperative load balancing game defined above there is a unique bargaining point and the bargaining solutions are determined by solving the following optimization problem:

$$\min_X [G(\lambda) \prod_{i=1}^n D_i(\beta_i)] \quad (3.26)$$

subject to the constraints (3.23) - (3.25).

Proof: In Appendix A. □

Theorem 3.4.2. For the cooperative load balancing game defined above the bargaining solution is determined by solving the following optimization problem:

$$\min_X \left[\sum_{i=1}^n \ln D_i(\beta_i) + \ln G(\lambda) \right] \quad (3.27)$$

subject to the constraints (3.23) - (3.25).

Proof: In Appendix A. □

Theorem 3.4.3. The solution to the optimization problem in Theorem 3.4.2 satisfies the relations

$$\begin{aligned} d_i(\beta_i) &\geq \alpha + g(\lambda), & \beta_i &= 0 & (i \in R_d), \\ d_i(\beta_i) &= \alpha + g(\lambda), & 0 < \beta_i &< \phi_i & (i \in R_a), \\ \alpha + g(\lambda) &\geq d_i(\beta_i) \geq \alpha, & \beta_i &= \phi_i & (i \in N), \\ d_i(\beta_i) &= \alpha, & \beta_i &> \phi_i & (i \in S), \end{aligned} \quad (3.28)$$

subject to the total flow constraint,

$$\sum_{i \in S} d_i^{-1}(\alpha) + \sum_{i \in N} \phi_i + \sum_{i \in R_a} d_i^{-1}(\alpha + g(\lambda)) = \Phi \quad (3.29)$$

where α is the Lagrange multiplier.

Proof: In Appendix A. □

Since obtaining a closed form solution for α from eq. (3.29) is not possible, we use a simple method such as a binary search to solve eq. (3.29) iteratively for α as in (Kim and Kameda 1992). Also, from Theorem 3.4.3, the following properties which are true in the optimal solution can be derived and their proofs are similar to those in (Kim and Kameda 1992).

Property 3.4.1.

$$\begin{aligned} d_i(0) &\geq \alpha + g(\lambda), & \text{iff } \beta_i &= 0, \\ d_i(\phi_i) &> \alpha + g(\lambda) > d_i(0), & \text{iff } 0 < \beta_i < \phi_i, \\ \alpha &\leq d_i(\phi_i) \leq \alpha + g(\lambda), & \text{iff } \beta_i &= \phi_i, \\ \alpha &> d_i(\phi_i), & \text{iff } \beta_i &> \phi_i. \end{aligned}$$

Property 3.4.2. If β is an optimal solution to the problem in Theorem 3.4.2, then we have

$$\begin{aligned} \beta_i &= 0, & i &\in R_d, \\ \beta_i &= d_i^{-1}(\alpha + g(\lambda)), & i &\in R_a, \\ \beta_i &= \phi_i, & i &\in N, \\ \beta_i &= d_i^{-1}(\alpha), & i &\in S. \end{aligned}$$

Property 3.4.3. If β is an optimal solution to the problem in Theorem 3.4.2, then we have $\lambda = \lambda_S = \lambda_R$, where

$$\begin{aligned} \lambda_S &= \sum_{i \in S} [d_i^{-1}(\alpha) - \phi_i], \\ \lambda_R &= \sum_{i \in R_d} \phi_i + \sum_{i \in R_a} [\phi_i - d_i^{-1}(\alpha + g(\lambda_S))]. \end{aligned}$$

Based on the above properties, we can have the following definitions (eqs. (3.30) - (3.35)) for an arbitrary $\alpha (\geq 0)$.

$$S(\alpha) = \{i | d_i(\phi_i) < \alpha\} \tag{3.30}$$

$$\lambda_S(\alpha) = \sum_{i \in S(\alpha)} [d_i^{-1}(\alpha) - \phi_i] \tag{3.31}$$

$$R_d(\alpha) = \{i | d_i(0) \geq \alpha + g(\lambda_S(\alpha))\} \tag{3.32}$$

$$R_a(\alpha) = \{i | d_i(\phi_i) > \alpha + g(\lambda_S(\alpha)) > d_i(0)\} \tag{3.33}$$

$$\lambda_R(\alpha) = \sum_{i \in R_d(\alpha)} \phi_i + \sum_{i \in R_a(\alpha)} [\phi_i - d_i^{-1}(\alpha + g(\lambda_S(\alpha)))] \tag{3.34}$$

$$N(\alpha) = \{i | \alpha \leq d_i(\phi_i) \leq \alpha + g(\lambda_S(\alpha))\} \quad (3.35)$$

Thus, if an optimal α is given, the node partitions in the optimal solution are characterized as

$$R_d = R_d(\alpha), R_a = R_a(\alpha), N = N(\alpha), S = S(\alpha) \quad (3.36)$$

and

$$\lambda = \lambda_S = \lambda_R = \lambda_S(\alpha) = \lambda_R(\alpha) \quad (3.37)$$

We now present an algorithm (CCOOP) for obtaining the Nash Bargaining Solution for our cooperative load balancing game.

CCOOP algorithm:

Input:

Node processing rates: $\mu_1, \mu_2, \dots, \mu_n$;

Node job arrival rates: $\phi_1, \phi_2, \dots, \phi_n$;

Mean communication time: t .

Output:

Load allocation to the nodes: $\beta_1, \beta_2, \dots, \beta_n$.

1. Initialization:

$$\beta_i \leftarrow \phi_i; i \in N; i = 1, \dots, n$$

2. Sort the computers such that $d_1(\phi_1) \leq d_2(\phi_2) \leq \dots, d_n(\phi_n)$.

If $d_1(\phi_1) + g(0) \geq d_n(\phi_n)$, then STOP (No load balancing is required).

3. Determine α (using a binary search):

$$a \leftarrow d_1(\phi_1)$$

$$b \leftarrow d_n(\phi_n)$$

while(1) do

$$\lambda_S(\alpha) \leftarrow 0$$

$$\lambda_R(\alpha) \leftarrow 0$$

$$\alpha \leftarrow \frac{a+b}{2}$$

Calculate: $S(\alpha), \lambda_S(\alpha), R_d(\alpha), R_a(\alpha)$, and $\lambda_R(\alpha)$ (eqs. (3.30) - (3.34))

in the order given for $i = 1, \dots, n$.

If $(|\lambda_S(\alpha) - \lambda_R(\alpha)| < \epsilon)$ **EXIT**

If $(\lambda_S(\alpha) > \lambda_R(\alpha))$

$b \leftarrow \alpha$

else

$a \leftarrow \alpha$

4. Determine the loads on the computers:

$\beta_i \leftarrow 0$, for $i \in R_d(\alpha)$

$\beta_i \leftarrow d_i^{-1}(\alpha + g(\lambda))$, for $i \in R_a(\alpha)$

$\beta_i \leftarrow d_i^{-1}(\alpha)$, for $i \in S(\alpha)$

$\beta_i \leftarrow \phi_i$, for $i \in N(\alpha)$ (eq. (3.35))

Remark 3.4.2. (i) In step 2, we STOP when the total (node + communication) time for a job to be transferred from a more powerful to a less powerful node exceeds the node time (node delay) on the less powerful node, if the network traffic equals 0. This means that a job will run faster on the ‘origin’ node than if transferred to a different node. (ii) The running time of this algorithm is $O(n \log n + n \log 1/\epsilon)$, where ϵ denotes the acceptable tolerance used for computing α in step 3 of the algorithm. (iii) This algorithm must be run periodically when the system parameters change in order to recompute a new load allocation.

3.5 Modeling Results

We modeled a computer system to evaluate the performance of our CCOOP scheme. The performance metrics used in our model are the *expected response time* and the *fairness index*. The *fairness index* (Jain 1991), is used to quantify the fairness of load balancing schemes. We study the impact of system utilization and heterogeneity on the performance of the proposed scheme. We also implemented the Overall Optimal Scheme (OPTIM) (Kim and Kameda 1992) and the Proportional Scheme (PROP) (Chow and Kohler 1979) for comparison. In the following we present and discuss the modeling results.

3.5.1 Effect of System Utilization

System utilization (ρ) represents the amount of load on the system and is defined as the ratio of the total arrival rate to the aggregate processing rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^n \mu_i} \quad (3.38)$$

We modeled a heterogeneous system consisting of 16 computers to study the effect of system utilization. The system has computers with four different processing rates. The system configuration is shown in Table 3.1.

Table 3.1: System configuration

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

The total job arrival rate in the system Φ is determined by the system utilization ρ and the aggregate processing rate of the system. We choose fixed values for the system utilization and determined the total job arrival rate Φ . The job arrival rate for each computer $\phi_i, i = 1, \dots, 16$ is determined from the total arrival rate as $\phi_i = q_i \Phi$, where the fractions q_i are given in Table 3.2. The mean communication time t is assumed to be 0.001 sec.

Table 3.2: Job arrival fractions q_i for each computer

Computer	1-2	3-6	7-11	12-14	15-16
q_i	0.01	0.02	0.04	0.1	0.2

In Figure 3.2, we present the expected response time of the system for different values of system utilization ranging from 10% to 90%. It can be seen that CCOOP performs as well as OPTIM for ρ ranging from 10% to 40% and is better than PROP for ρ ranging from 50% to 60%. CCOOP approaches PROP at high system utilization.

In Figure 3.3, we present the fairness index for different values of system utilization. The CCOOP scheme has a fairness index of almost 1 for any system utilization. The fairness index of

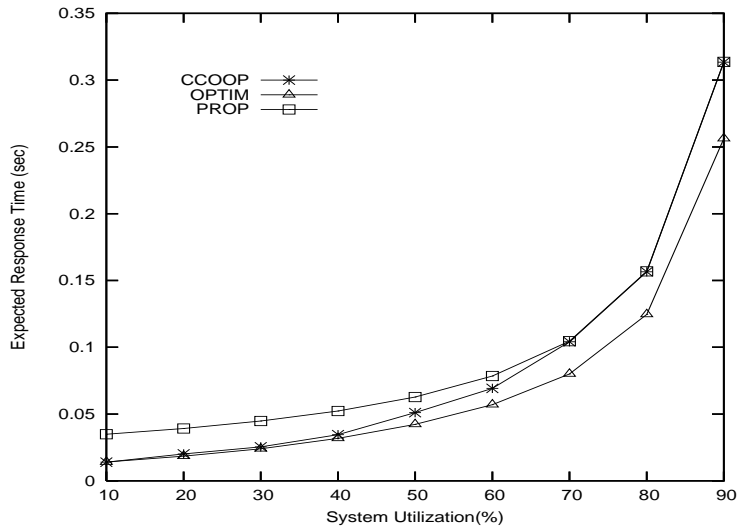


Figure 3.2: System Utilization vs Expected Response Time

OPTIM drops from 1 at low load to 0.89 at high load and PROP maintains a fairness index of 0.73 over the whole range of system loads.

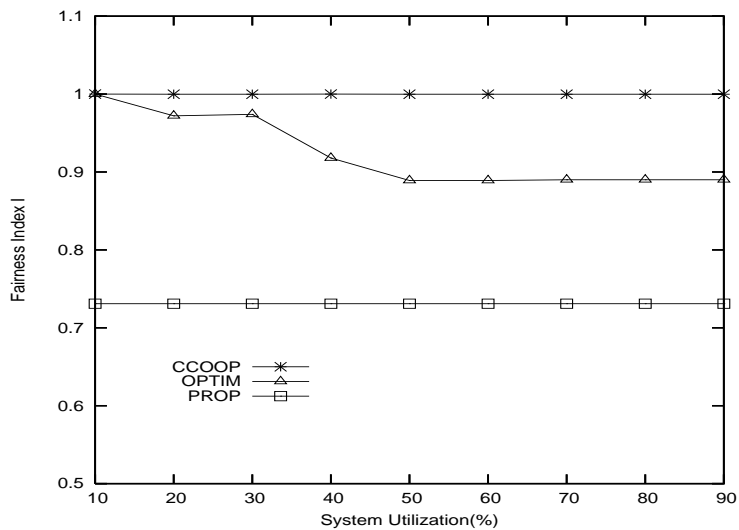


Figure 3.3: System Utilization vs Fairness Index

3.5.2 Effect of Heterogeneity

In this section, we study the effect of heterogeneity on the performance of load balancing schemes. One of the common measures of heterogeneity is the *speed skewness* (Tang and Chanson 2000). We study the effectiveness of load balancing schemes by varying the speed skewness.

We modeled a heterogeneous system of 16 computers (2 fast and 14 slow) to study the effect of heterogeneity. The slow computers have a relative processing rate of 1 and the relative processing

rate of the fast computers is varied from 1 (homogeneous system) to 20 (highly heterogeneous system). The system utilization was kept constant ($\rho = 60\%$) and the mean communication time t is assumed to be 0.001 sec. In Table 3.3, we present the processing rates (μ_i jobs/sec) of the computers in the systems and the total arrival rates (Φ) for some of the cases. C1 and C2 represent the fast computers and C3 through C16 represent the slow computers.

Figure 3.4 shows the effect of speed skewness on the expected response time. For low skewness, CCOOP behaves like the PROP. But, as the skewness increases, the performance of CCOOP approaches to that of OPTIM which means that in highly heterogeneous systems CCOOP is very effective.

Figure 3.5 shows the effect of speed skewness on the fairness index. It can be observed that CCOOP has a fairness index of almost 1 over all range of speed skewness. The fairness index of OPTIM and PROP falls from 1 at low skewness to 0.92 and 0.88 respectively at high skewness.

Table 3.3: System parameters

Speed skewness	1	4	8	12	16	20
μ_i of C1,C2	10	40	80	120	160	200
μ_i of C3-C16	10	10	10	10	10	10
Φ (jobs/sec)	96	132	180	228	276	324

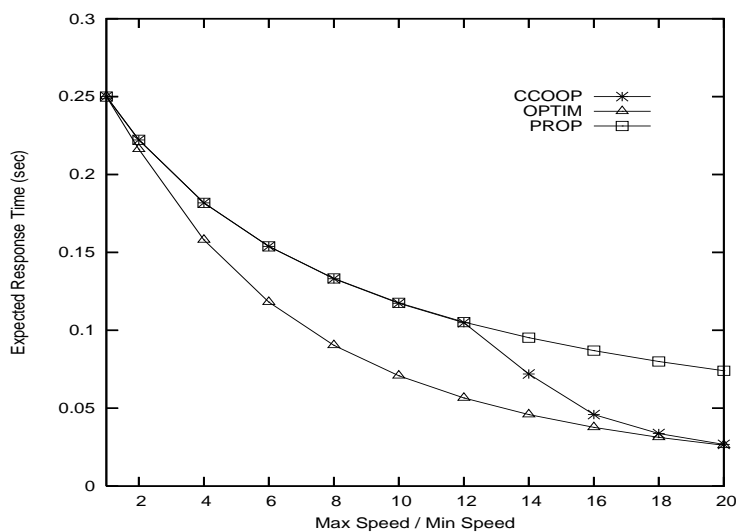


Figure 3.4: Heterogeneity vs Expected Response Time

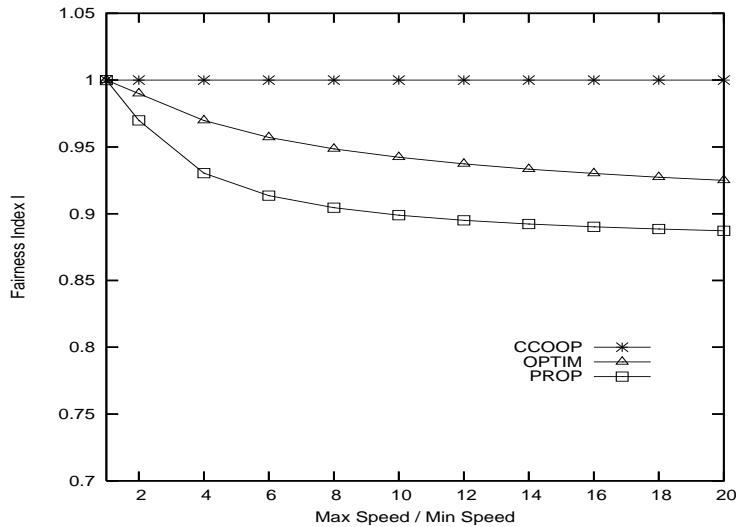


Figure 3.5: Heterogeneity vs Fairness Index

3.5.3 Effect of Communication Time

In Figure 3.6, we present the effect of mean communication time (t) on the performance of CCOOP. The system configuration used is the same as in Table 3.1. We also implemented COOP scheme (Grosu, Chronopoulos, and Leung 2002) for comparison. It can be observed that as t increases (from 0.001 sec to 0.01 sec), CCOOP outperforms COOP. Although the figure shows plots for only a few values, the performance decrease of COOP compared to CCOOP is more for any value of $t > 0$. This is because CCOOP tries to load balance by taking the communication delays into account whereas COOP does not.

Figure 3.7 shows the effect of communication time on the performance COOP and CCOOP for 50% and 70% system utilizations. It can be observed that, as t increases, the performance degradation of COOP is more compared to that of CCOOP (for example, for $t = 0.01sec$, there is a 17% decrease in the performance of COOP at 50% system utilization when compared to that of CCOOP). Thus, CCOOP can show significant performance improvement over COOP in highly congested systems.

3.6 Summary

In this chapter, we presented a game theoretic approach to solve the static load balancing problem in single-class job distributed systems where the computers are connected by a communication network. We used a cooperative game to model the load balancing problem. Our solution is based on the

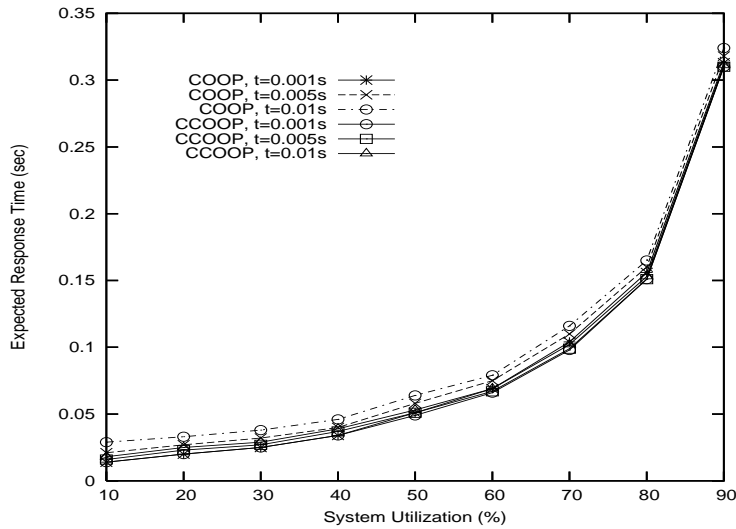


Figure 3.6: Effect of Communication Time

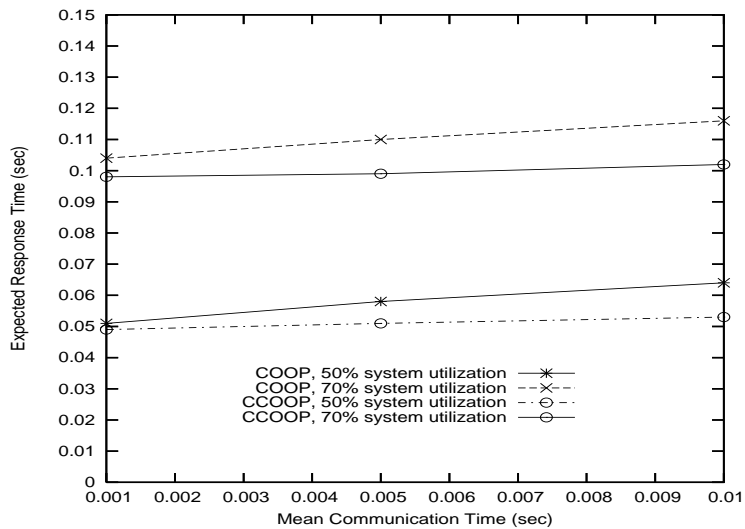


Figure 3.7: Communication Time vs Expected Response Time

NBS which provides a Pareto optimal and fair solution to the distributed system. For the proposed cooperative load balancing game we derived an algorithm for computing the NBS. Using a computer model, the performance of our scheme is compared with that of other existing schemes. We showed that our CCOOP scheme is near OPTIM in terms of performance, but, unlike OPTIM, it also provides fairness to the individual jobs.

In the next chapter, we extend the single-class job distributed system model of this chapter to a multi-class job grid system model and propose price-based job allocation schemes for computational grids.

CHAPTER 4: Job Allocation Schemes for Computational Grids

4.1 Introduction

Grid computing (Foster and Kesselman 2004) is an important developing computing infrastructure which is a conglomeration of computing resources connected by a network, to form a distributed system used for solving complex scientific, engineering and commercial problems. This system tries to solve these problems or applications by allocating the idle computing resources over a network or the internet commonly known as the *computational grid*. These computational resources have different owners who can be enabled by an automated negotiation mechanism by the grid controllers and this can be viewed as a market-oriented grid (Chao, Anane, Chen, and Gatward 2002). This market-oriented grid is concerned with a particular type of resource like the computational power or storage for which these negotiations are made.

A job or an application usually requires the resources from more than one owner. So, these grid computing systems should be able to assign the jobs of various users to the different owned resources efficiently and utilize the resources of unused devices, commonly known as the automated *load balancing/job scheduling* problem. The purpose of load balancing is to improve the performance of the grid system through an appropriate distribution of the user's application load.

4.1.1 Related Work and Our Contribution

Ghosh *et al.* studied price-based job allocation for mobile grid systems ((Ghosh, Roy, Basu, and Das 2004) and references therein). However, they considered a very specialized grid system model where the jobs arrive to a single server which then allocates them to the mobile devices. This is a single-server many-computer scheduling algorithm. Also, jobs in their system belong to only one class and their scheme tries to provide a system-optimal solution. So, some jobs may be charged more price (amount that has to be paid for using the resources) for execution than others, which is unfair. There are some studies for job allocation in multi-class job distributed systems that try to provide fairness to the users ((Kameda, Li, Kim, and Zhang 1997; Grosu and Chronopoulos 2005) and references therein). However, they did not include pricing in the model.

In this chapter (Penmatsa and Chronopoulos 2005), we propose two static price-based job

allocation schemes for multi-class job grid systems. We use an existing pricing model to obtain the prices that the grid users has to pay to the grid resource owners for using their resources. We also assume that the jobs assigned to a computer by a grid server are processed completely by that computer and are not transferred any further (the communication subsystem is not taken into account). The two schemes differ in their objective. (i) The global optimal job allocation scheme we propose tries to minimize the expected price (or cost) for the entire grid community (*i.e.*, grid users) to provide a system-optimal solution. The problem is formulated as a cost minimization problem. This is an extension of a global optimal load balancing scheme for distributed systems (Kameda, Li, Kim, and Zhang 1997). (ii) The user optimal job allocation scheme we propose tries to provide a fair solution *i.e.*, the expected price (or cost) for all the grid users is almost the same independent of the computers allocated for the execution of their jobs. This is an extension of a non-cooperative load balancing scheme for distributed systems (Grosu and Chronopoulos 2005). Each grid server (acting on behalf of a user) tries to optimize its objective function (minimizing the price) independently of the others and they all eventually reach an equilibrium. This situation can be viewed as a non-cooperative game among the servers. The equilibrium is called *Nash equilibrium* and is obtained by a distributed non-cooperative policy. In general, jobs from a grid user will be dealt by a local server and so this scheme is favorable to the individual users but not to the entire system.

4.1.2 Chapter Organization

The rest of the chapter is structured as follows. In Section 4.2, we present the pricing model. In Section 4.3, we present the grid system model and formulate the problem. In Section 4.4, we describe our job allocation schemes and derive job allocation algorithms. The performance of the proposed schemes is evaluated in Section 4.5. A summary of the chapter is provided in Section 4.6.

4.2 Pricing Model

A grid system is a collection of grid servers and computers (*i.e.*, resources). These servers try to find the resources on which the jobs from various users can be executed. The negotiation between these two entities is formulated as an incomplete information alternating-offer non-cooperative bargaining game in (Owen 1982; Ghosh, Roy, Basu, and Das 2004; Winoto, McCalla, and Vassileva 2002)

with the grid servers playing on behalf of the grid users. Similar economic models based on game theory are proposed in (Buyya, Abramson, Giddy, and Stockinger 2002). The two players (servers and computers) have no idea of each other's reserved valuations (Owen 1982), *i.e.*, the maximum offered price for the server (acting as the buyer of resource) and the minimum expected price for the computers (acting as the seller of the resource). The server has to play an independent game with each computer associated with it to form the price per unit resource vector, p_j . So, in a system with m servers and n computers at time t , we have $m \times n$ bargaining games as shown in Figure 4.1.

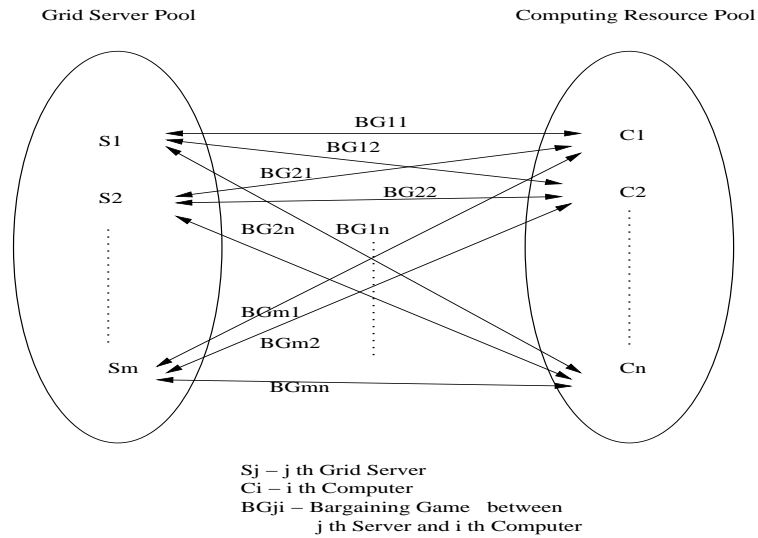


Figure 4.1: Bargaining game mapping between the grid servers and computers

Both the players try to maximize their utility functions and so the game reduces to the case of dividing the difference of maximum buying price offered by the grid community and minimum selling price expected by the computers.

The bargaining protocol is as follows: One of the players starts the game. If the server starts the game, it proposes an offer which will be much less than its own reserved valuation. If the offered price \geq the computer's standard price with highest expected surplus, then the computer accepts the offer. Else, it makes a counter offer. If this counter offer \leq the server's standard price with the highest expected surplus, it accepts. Else the server counter offers again. This procedure continues until an agreement is reached.

At each step, the expected surplus of each player is based on the probability of acceptance, breakdown or counter-offer of the other player. In general, they are given by: *Expected Utility* = $E[\text{Surplus}] = (\text{reserved valuation of } x - \text{standard price of } x) \times \text{probability}(\text{standard price})$ (Winoto, Mc-

Calla, and Vassileva 2002), where x stands for the grid server or the computer, $probability(standardprice)$ is the probability that the standard price will be accepted by the other player as predicted by itself and the standard price represents the different offered prices used by the players to compute their expected surplus. Also, at each step, if an offer is rejected, then the players will update (*i.e.*, reduce) the $probability(standardprice)$ which monotonically decreases as the alternatives come closer to their reserved valuations where it is more likely to be accepted by the opponent (Larson and Sandholm 2002).

We implemented this pricing model based on the assumptions given in (Ghosh, Roy, Basu, and Das 2004). Figures 4.2 and 4.3 show the expected surplus (profit) earned by the server and the computer against their various offered prices with time. As the time increases, the expected surplus gradually decreases, which makes both the players offer prices which are much closer to their reserved valuation and which helps the game to converge.

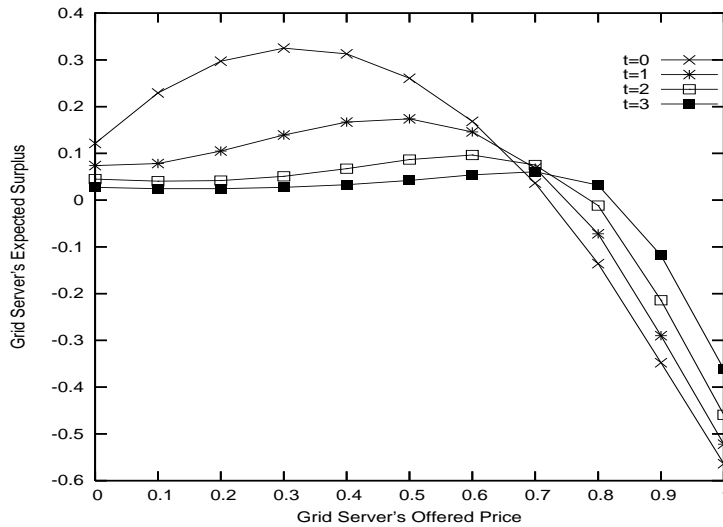


Figure 4.2: Expected surplus of the Grid server vs Offered prices

4.3 Grid System Model

We consider a grid system model with many servers and computers as shown in Figure 4.4. The system has m grid servers and n computers. The grid controller acting on behalf of the grid community (which consists of the users) assigns jobs to the grid servers from different users with a total job arrival rate of Φ . Let the job arrival rate at each server be ϕ_j . Hence, $\Phi = \sum_{j=1}^m \phi_j$. We assume that all the arriving jobs are of the same size. Each computer is modeled as an M/M/1 queuing system (*i.e.*,

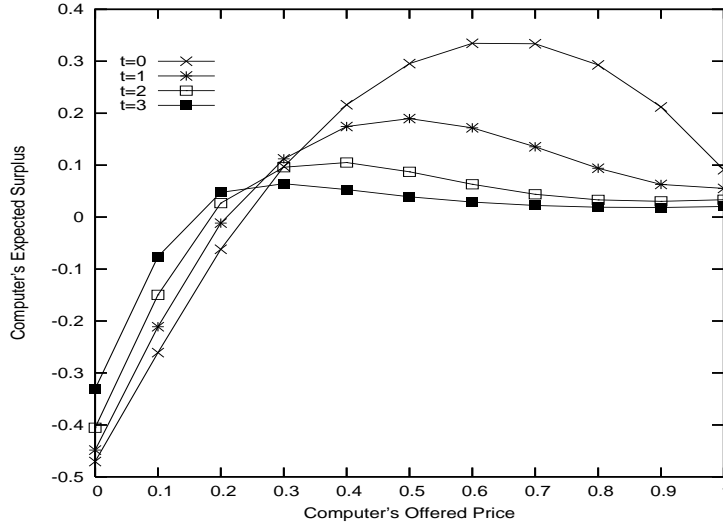


Figure 4.3: Expected surplus of the Computer vs Offered prices

Poisson arrivals and exponentially distributed processing times) (Kleinrock 1975) and is characterized by its average processing rate μ_i , $i = 1, \dots, n$. The total job arrival rate Φ must be less than the aggregate processing rate of the system (*i.e.*, $\Phi < \sum_{i=1}^n \mu_i$). Each server j keeps track of the price per unit resource p_{ji} (the bargaining game is played prior to the job allocation) and the processing rate μ_i of the i^{th} computer. Since each grid user will have a different reserved valuation, the p_{ji} depends on the user on whose behalf the server j plays the game with the i^{th} computer and so the server has to maintain separate price vectors for each grid user.

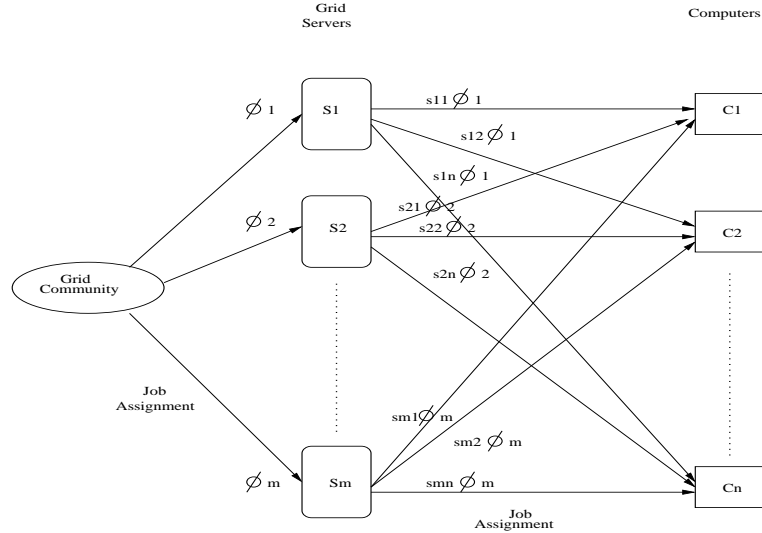


Figure 4.4: Grid System Model

Based on the proposed scheme, we find the load fractions (s_{ji}) of each server j ($j = 1, \dots, m$) that are assigned to computer i ($\sum_{i=1}^n s_{ji} = 1$ and $0 \leq s_{ji} \leq 1$, $i = 1, \dots, n$) such that the expected

cost for all the jobs in the system or the expected cost for the local jobs of the servers is minimized. In the following we present the notations and define the problem.

Let s_{ji} be the fraction of workload (jobs) that server j sends to computer i . Thus, $\mathbf{s}_j = (s_{j1}, s_{j2}, \dots, s_{jn})$ denotes the workload fractions of server j and the vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_m)$ denotes the load fractions of all the servers.

Since each computer is modeled as an M/M/1 queuing system, the expected response time at computer i is given by (the model is derived from queuing theory similar to Chapter 3):

$$F_i(\mathbf{s}) = \frac{1}{\mu_i - \sum_{j=1}^m s_{ji}\phi_j} \quad (4.39)$$

Thus the overall expected cost of server j is given by:

$$D_j(\mathbf{s}) = \sum_{i=1}^n k_i p_{ji} s_{ji} F_i(\mathbf{s}) = \sum_{i=1}^n \frac{k_i p_{ji} s_{ji}}{\mu_i - \sum_{k=1}^m s_{ki}\phi_k} \quad (4.40)$$

and the overall expected cost of the system (*i.e.*, of all the servers) is given by:

$$D(\mathbf{s}) = \frac{1}{\Phi} \sum_{j=1}^m \phi_j D_j(\mathbf{s}) \quad (4.41)$$

which is equivalent to

$$D(\mathbf{s}) = \frac{1}{\Phi} \sum_{j=1}^m \sum_{i=1}^n \frac{k_i p_{ji} \phi_j s_{ji}}{\mu_i - \sum_{k=1}^m s_{ki}\phi_k} \quad (4.42)$$

subject to the constraints:

$$s_{ji} \geq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, m \quad (4.43)$$

$$\sum_{i=1}^n s_{ji} = 1, \quad j = 1, \dots, m \quad (4.44)$$

$$\sum_{j=1}^m s_{ji}\phi_j < \mu_i, \quad i = 1, \dots, n \quad (4.45)$$

where k_i is assumed to be a constant which maps the response time to the amount of resources consumed at node i and p_{ji} is the agreed price as a result of the bargaining game between server j

and computer i .

Based on the above, we propose two job allocation schemes, (i) GOSP which tries to minimize the expected cost for all the jobs in the system (*i.e.*, jobs at all the servers) to provide a system-optimal solution, and (ii) NASHP which tries to minimize the expected cost of all the jobs at each server independently of the others to provide a user-optimal (or fair) solution. We describe these two schemes in the next section.

4.4 Price-based Job Allocation Schemes

4.4.1 Global Optimal Scheme with Pricing (GOSP)

The load fractions (\mathbf{s}) are obtained by solving the non-linear optimization problem $D(\mathbf{s})$ (eq. (4.42)) which gives the optimum expected cost of the system. To find the solution, the scheme finds the load fractions of each server by taking into account the load on each computer due to the other server allocations. Let $\mu_i^j = \mu_i - \sum_{k=1, k \neq j}^m s_{ki} \phi_k$ be the *available processing rate* at computer i as seen by server j .

Theorem 4.4.1. Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geq \mu_2^j \geq \dots \geq \mu_n^j$), the load fractions for server j are given by:

$$s_{ji} = \begin{cases} \frac{1}{\phi_j} \left(\mu_i^j - \sqrt{k_i p_{ji} \mu_i} \frac{\sum_{k=1}^{c_j} \mu_k^j - \phi_j}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k}} \right) & \text{if } 1 \leq i < c_j \\ 0 & \text{if } c_j \leq i \leq n \end{cases} \quad (4.46)$$

where c_j is the minimum index that satisfies the inequality:

$$\mu_{c_j}^j \leq \frac{\sqrt{k_{c_j} p_{j c_j} \mu_{c_j}} (\sum_{k=1}^{c_j} \mu_k^j - \phi_j)}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k}} \quad (4.47)$$

Proof: In Appendix B. □

Based on the above theorem we derived the following algorithm for determining server j 's best load fractions.

BEST-FRACTIONS ($\mu_1^j, \dots, \mu_n^j, \phi_j, p_{j1}, \dots, p_{jn}, k_1, \dots, k_n$)

Input: Available processing rates: $\mu_1^j, \mu_2^j, \dots, \mu_n^j$.

Total arrival rate: ϕ_j .

The price per unit resource vector: $p_{j1}, p_{j2}, \dots, p_{jn}$.

The constants vector: k_1, k_2, \dots, k_n .

Output: Load fractions: $s_{j1}, s_{j2}, \dots, s_{jn}$.

1. Sort the computers in decreasing order of

$$\left(\frac{\mu_1^j}{\sqrt{\mu_1 k_1 p_{j1}}} \geq \frac{\mu_2^j}{\sqrt{\mu_2 k_2 p_{j2}}} \geq \dots \geq \frac{\mu_n^j}{\sqrt{\mu_n k_n p_{jn}}} \right);$$

$$2. t \leftarrow \frac{\sum_{i=1}^n \mu_i^j - \phi_j}{\sum_{i=1}^n \sqrt{\mu_i p_{ji} k_i}}$$

3. **while** ($t \geq \frac{\mu_n^j}{\sqrt{\mu_n k_n p_{jn}}}$) **do**

$$s_{jn} \leftarrow 0$$

$$n \leftarrow n - 1$$

$$t \leftarrow \frac{\sum_{i=1}^n \mu_i^j - \phi_j}{\sum_{i=1}^n \sqrt{\mu_i p_{ji} k_i}}$$

4. **for** $i = 1, \dots, n$ **do**

$$s_{ji} \leftarrow \frac{1}{\phi_j} \left(\mu_i^j - t \sqrt{\mu_i p_{ji} k_i} \right)$$

The following theorem proves the correctness of this algorithm.

Theorem 4.4.2. The load fractions $\{s_{j1}, s_{j2}, \dots, s_{jn}\}$ computed by the algorithm BEST-FRACTIONS, solves the optimization problem $D(\mathbf{s})$ and are the optimal fractions for server j .

Proof: In Appendix B. □

To compute the optimal load fractions of all the servers, there should be some communication between them in order to obtain the load information from the other servers and compute the μ_i^j 's.

Based on the BEST-FRACTIONS algorithm presented above, we devise the following iterative algorithm where each server updates from time to time its load fractions taking into account the existing load fractions of the other servers in a round-robin fashion.

We use the following notations:

j - the server number;

l - the iteration number;

$\mathbf{s}_j^{(l)}$ - the load fractions of server j computed at iteration l ;

$D_j^{(l)}$ - server j 's expected price at iteration l ;

ϵ - a properly chosen acceptance tolerance;

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

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

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

GOSP job allocation algorithm:

User j , ($j = 1, \dots, m$) executes:

1. Initialization:

$\mathbf{s}_j^{(0)} \leftarrow \mathbf{0}$;

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

$l \leftarrow 0$;

$norm \leftarrow 1$;

$sum \leftarrow 0$;

$tag \leftarrow \text{CONTINUE}$;

$left = [(j - 2) \bmod m] + 1$;

$right = [j \bmod m] + 1$;

2. **while** (1) **do**

if($j = 1$) {server 1}

if ($l \neq 0$)

Recv($left, (norm, l, tag)$);

if ($norm < \epsilon$)

Send($right, (norm, l, \text{STOP})$);

exit;

$sum \leftarrow 0$;

$l \leftarrow l + 1$;

else {the other servers}

Recv($left, (sum, l, tag)$);

```

if ( $tag = \text{STOP}$ )
    if ( $j \neq m$ )
        Send( $right, (sum, l, \text{STOP})$ );
    exit;
for  $i = 1, \dots, n$  do
    Obtain  $\mu_i^j$  by inspecting the run queue of each computer
    ( $\mu_i^j \leftarrow \mu_i - \sum_{k=1, k \neq j}^m s_{ki} \phi_k$ );
 $\mathbf{s}_j^{(l)} \leftarrow \text{BEST-FRACTIONS}(\mu_1^j, \dots, \mu_n^j, \phi_j, p_{j1}, \dots, p_{jn}, k_1, \dots, k_n)$ ;
    Compute  $D_j^{(l)}$ ;
     $sum \leftarrow sum + |D_j^{(l-1)} - D_j^{(l)}|$ ;
    Send( $right, (sum, l, \text{CONTINUE})$ );
endwhile

```

This iterative algorithm can be implemented on the distributed system and can be restarted periodically or when the system parameters are changed. Once the accepted tolerance is reached, the servers will continue to use the same load fractions and the system operates at the optimal cost. The running time of each iteration is $O(mn \log n + mn \log(1/\epsilon))$. This Global Optimal Scheme with Pricing (GOSP) minimizes the expected cost over all the jobs executed by the Grid system.

4.4.2 Nash Scheme with Pricing (NASHP)

We consider the NASH algorithm for load balancing in distributed systems (Grosu and Chronopoulos 2002) and modify it to include pricing. In this scheme each server tries to minimize the total cost of its jobs independently of the others. The load fractions are obtained by formulating the problem as a non-cooperative game among the servers. The goal of server j is to find a feasible job allocation strategy \mathbf{s}_j such that $D_j(\mathbf{s})$ (eq. (4.40)) is minimized.

The best allocation strategy of server j which is the solution of (eq. (4.40)) is given by the following theorem.

Theorem 4.4.3. Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geq \mu_2^j \geq \dots \geq \mu_n^j$), the solution \mathbf{s}_j of the optimization problem $D_j(\mathbf{s})$ is given

by:

$$s_{ji} = \begin{cases} \frac{1}{\phi_j} \left(\mu_i^j - \sqrt{k_i p_{ji} \mu_i^j \frac{\sum_{k=1}^{c_j} \mu_k^j - \phi_j}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k^j}}} \right) & \text{if } 1 \leq i < c_j \\ 0 & \text{if } c_j \leq i \leq n \end{cases} \quad (4.48)$$

where c_j is the minimum index that satisfies the inequality:

$$\sqrt{\mu_{c_j}^j} \leq \frac{\sqrt{k_{c_j} p_{jc_j}} (\sum_{k=1}^{c_j} \mu_k^j - \phi_j)}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k^j}} \quad (4.49)$$

Proof: Similar to that of GOSP. □

Based on the above theorem we have the **BEST-REPLY** algorithm similar to that of the BEST-FRACTIONS for determining server j 's best strategy. The computation of Nash equilibrium may require some communication between the servers. Each server updates from time to time its job allocation strategy by computing the best response against the existing job allocation strategies of the other servers. Based on the BEST-REPLY algorithm we devised a greedy best reply algorithm for computing the Nash equilibrium for the non-cooperative job allocation scheme which is similar to the GOSP job allocation algorithm by replacing the procedure call to BEST-FRACTIONS by BEST-REPLY.

4.5 Modeling Results

We used a grid model to evaluate the performance of our GOSP and NASHP schemes. The performance metrics used in our model are the *expected response time* and the *fairness index*. The *fairness index* (Grosu and Chronopoulos 2002) (defined from the servers' perspective),

$$I(\mathbf{C}) = \frac{[\sum_{j=1}^m C_j]^2}{m \sum_{j=1}^m C_j^2} \quad (4.50)$$

is used to quantify the fairness of job allocation schemes. Here the parameter \mathbf{C} is the vector $\mathbf{C} = (C_1, C_2, \dots, C_m)$ where C_j is the expected cost of server j 's jobs. This index is a measure of the 'equality' of servers' total expected cost. If all the servers have the same total expected cost then $I = 1$ and the system is 100% fair to all servers and it is cost-balanced. If the differences on C_j increase, I decreases and the job allocation scheme favors only some servers. If the cost is

Table 4.1: System configuration

Relative μ_i	1	2	3	4	5	7	8	10
#computers	7	6	5	4	3	3	2	2
μ_i (jobs/sec)	10	20	30	40	50	70	80	100
k_i	1	2	3	4	5	6	7	8

proportional to the load, then cost-balanced is also load-balanced.

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

4.5.1 Effect of System Utilization

To study the effect of system utilization we modeled a heterogeneous system consisting of 32 computers with eight different processing rates. This system is shared by 20 servers. The price vector p_j for each server is based on the alternating offer bargaining game previously described. In Table 4.1, we present the system configuration. The first row contains the relative processing rates of each of the eight 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 third row shows the processing rate of each computer type in the system. The last row shows the values for k_i , the constant which maps the response time at the computer i to the amount of resources consumed at i (Ghosh, Roy, Basu, and Das 2004).

The total job arrival rate in the system Φ is determined by the system utilization ρ and the aggregate processing rate of the system. *System utilization* (ρ) is defined as the ratio of the total arrival rate to the aggregate processing rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^n \mu_i} \quad (4.51)$$

We choose fixed values for the system utilization and determined the total job arrival rate Φ .

Figure 4.5 shows the plots for total price that the grid user has to pay as a function of system

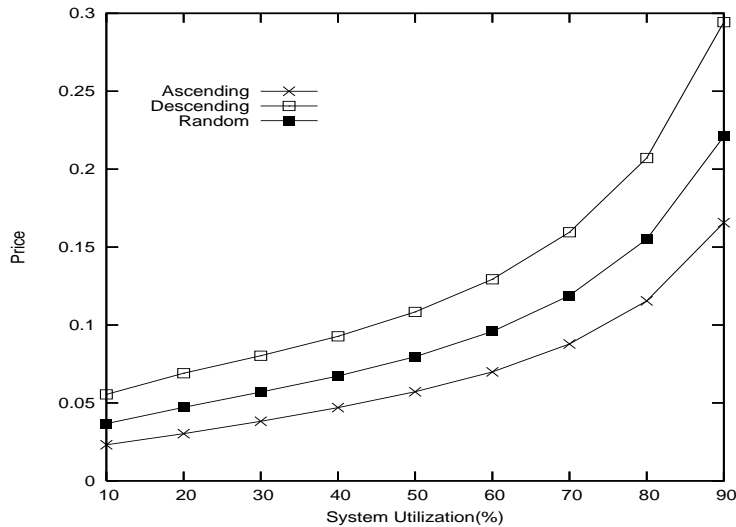


Figure 4.5: System Utilization vs Expected Price (or Cost)

utilization based on GOSP. The price increases with system utilization because the higher the ρ , the more the load on the computers and so the higher the expected response time and the cost. Three curves are shown corresponding to random, strictly decreasing and strictly increasing price vector (the computers are initially numbered in decreasing order of their processing rates). The random price vector is the one obtained by the pricing strategy described above and the corresponding curve lies between that for the ascending and the descending price vector cases. This is because, if the faster computers charge less (in the case of price vector in ascending order), then they will get the bulk of the work resulting in lower overall expected response time and subsequently lower total expected price for the grid user. Similarly, if the faster devices charge more (in the case of price vector in descending order), then they will get fewer jobs resulting in greater overall expected response time and subsequently greater expected price for the grid user.

In Figures 4.6 and 4.7, we present the expected response time of the system and the fairness index for different values of system utilization (ranging from 10% to 90%).

It can be seen that for different system loads, the GOSP scheme which minimizes the cost of the entire system performs better than the NASHP scheme where each server minimizes its own cost. But, GOSP whose objective is to reduce the overall cost of the grid community (users) is unfair (fairness index falls to as low as 0.68 for high system loads) whereas NASHP has a fairness index of almost 1 for any system load, which means that it is fair to each server and thus to each user.

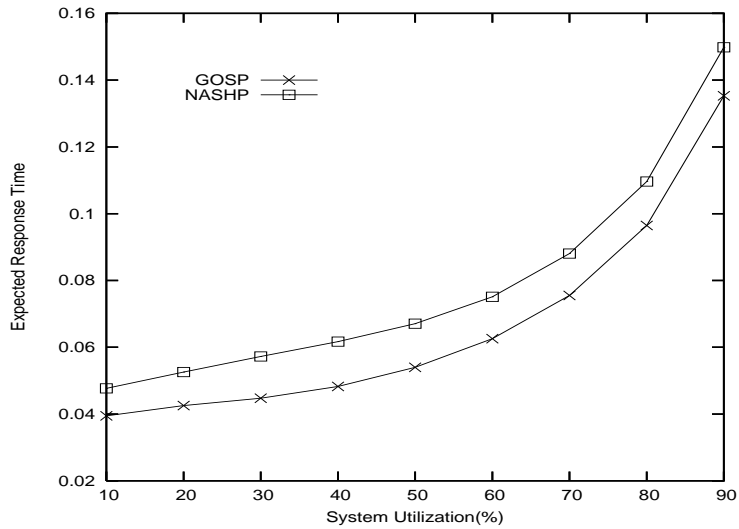


Figure 4.6: System Utilization vs Expected Response Time

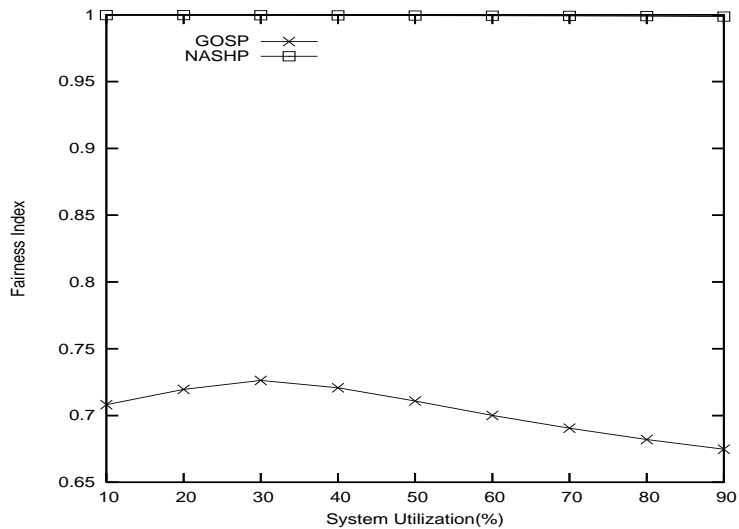


Figure 4.7: System Utilization vs Fairness Index

4.5.2 Effect of Heterogeneity

In a grid, 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* (Tang and Chanson 2000) which is defined as the ratio of maximum processing rate to minimum processing rate of the grid computers.

In this section, we investigate the effectiveness of load balancing schemes by varying the speed skewness. We modeled a system of 32 heterogeneous computers: 4 fast and 28 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 $\rho = 60\%$.

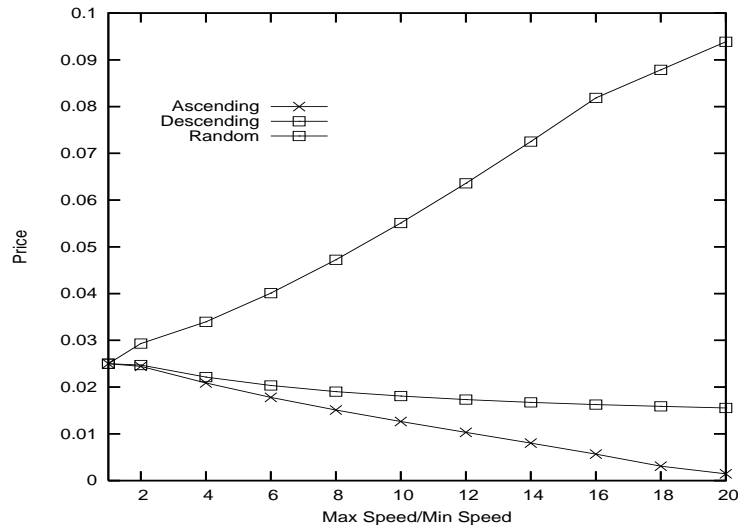


Figure 4.8: Heterogeneity vs Expected Price

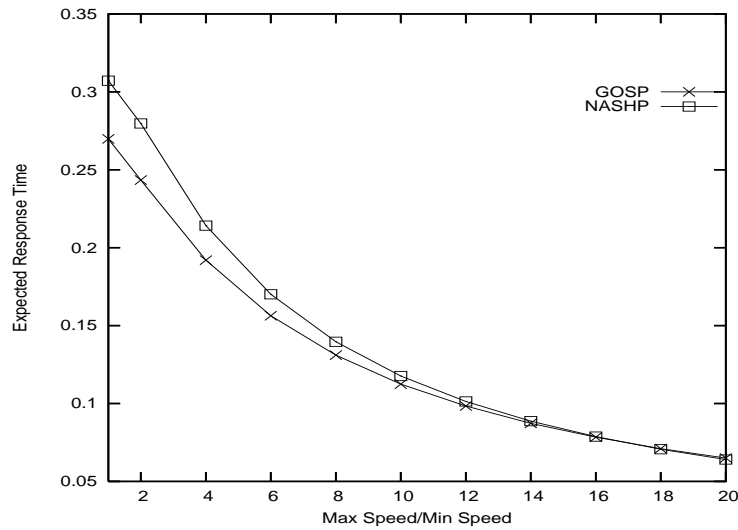


Figure 4.9: Heterogeneity vs Expected Response Time

Figure 4.8 shows the plots for total price that the grid user has to pay with increasing speed skewness for three different price vectors based on GOSP. The expected price for the random price vector lies between that for the ascending and descending price vectors for similar reasons as discussed before. Figure 4.9 plots the overall expected response time with increasing speed skewness. The total expected response time decreases for both the schemes with an increase in the relative processing speed of the fast computers and GOSP performs better than the NASHP at all times and at high skewness, both perform equally well.

4.6 Summary

In this chapter, we proposed two price-based job allocation schemes for multi-class job grid systems. These schemes are formulated as a constraint minimization problem and as a non-cooperative game respectively. Two algorithms to compute the optimal load (job) fractions for the grid servers are presented. The first scheme (GOSP) tries to minimize the expected cost for the entire grid system and so it is advantageous when the system optimum is required. However, it is not fair to the servers and so to the users (because, the servers act on behalf of the users). The second scheme (NASHP) tries to minimize the expected cost for each server. This is fair to the servers and therefore it is fair to the users.

In the next chapter, we extend the multi-class job grid system model considered in this chapter to include the communication subsystem and propose a price-based user-optimal job allocation scheme.

CHAPTER 5: Job Allocation Scheme for Computational Grids with Communication Costs

5.1 Introduction

A *Grid* (Foster and Kesselman 2004) is a distributed computing infrastructure which uses the resources of many independent computers connected by a network to solve large-scale computation problems. It can also be viewed as a collection of clusters where the computers in a cluster are connected by a fast local area network. The grid controllers try to solve the computational problems of the grid users by allocating them to the idle computing resources. These resources which may have different owners can be enabled by an automated negotiation mechanism by the grid controllers.

5.1.1 Related Work and Our Contribution

Past game theory and other related works on job allocation and load balancing in general distributed systems and also in grid systems can be found in ((Ghosh, Roy, Basu, and Das 2004; Penmatsa and Chronopoulos 2005; Kwok, Song, and Hwang 2005; Ghosh, Basu, and Das 2005; Grosu and Das 2004; Yu and Robertazzi 2003; Inoie, Kameda, and Touati 2004; Altman, Kameda, and Hosokawa 2002) and references there in). However, these works did not take the communication-subsystem or fairness-to-the-users into account. In this chapter (Penmatsa and Chronopoulos 2006b), we consider a grid model where the computers are connected by a communication network. Prior to any job scheduling, the grid controllers submit the users jobs to the various computers based on their resource availability. Then, job scheduling is achieved by transferring some jobs from a heavily loaded computer to a lightly loaded one.

Here, we propose a price-based job allocation scheme whose objective is to minimize the cost of the individual grid users. Since the grid controllers act on behalf of the grid users, we use the term ‘grid user’ instead of ‘grid controller’ to avoid ambiguity. The scheme is formulated as a non-cooperative game among the grid users who try to minimize the expected cost of their own jobs. The main goal of our job allocation scheme is to provide fairness to all the users, *i.e.*, all the users should have the same expected cost independent of the allocated computer.

5.1.2 Pricing Model

In a grid system, to allocate a job to a resource, an agreement should be made between the grid user and the resource owner. We use a pricing model based on the bargaining game theory to obtain the prices charged by the resource owners (computers) (Ghosh, Roy, Basu, and Das 2004). Similar economic models are studied in (Buyya, Abramson, Giddy, and Stockinger 2002; Winoto, McCalla, and Vassileva 2002; Larson and Sandholm 2002; Osborne and Rubinstein 1990; Buyya, Abramson, and Venugopal 2005; Buyya, Murshed, Abramson, and Venugopal 2005). The two players (users and computers) negotiate until an agreement is reached. Each user has to play an independent game with each computer to obtain its price per unit resource on that computer. We implemented this pricing model and used the prices obtained for job allocation.

5.1.3 Chapter Organization

The rest of the chapter is organized as follows. In Section 5.2, we present the grid system model. In Section 5.3, we formulate the job allocation problem as a non-cooperative game among the grid users. In Section 5.4, the performance of our user-optimal job allocation scheme is compared with a system-optimal job allocation scheme by modeling a grid system. A summary of the chapter is provided in Section 5.5.

5.2 Grid System Model

We consider a grid system model as shown in Figure 5.1. The system has n nodes (computers) connected by a communication network and is shared by m users. The nodes could be either single computers or clusters (of several computers). The nodes and the communication network are assumed to be product-form queuing network models (Jain 1991). In the following, we use 'computer' and 'node' interchangeably. The terminology and assumptions used similar to (Kim and Kameda 1990) are as follows:

- ϕ_i^j : Job arrival rate of user j to computer i .
- ϕ^j : Total job arrival rate of user j .
- $\phi^j = \sum_{i=1}^n \phi_i^j$.

- Φ : Total job arrival rate of the system. All the jobs in the system are assumed to be of the same size.
- $\Phi = \sum_{j=1}^m \phi^j$.
- μ_i : Service rate of computer i .
- β_i^j : Job processing rate (load) of user j at computer i .
- $\beta_i = [\beta_i^1, \beta_i^2, \dots, \beta_i^m]^T$; $\beta = [\beta_1, \beta_2, \dots, \beta_n]^T$.
- $\beta^k = [\beta_1^k, \beta_2^k, \dots, \beta_n^k]^T$.
- x_{rs}^j : Job flow rate of user j from computer r to computer s .
- λ^j : Job traffic through the network of user j .
- $\lambda^j = \sum_{r=1}^n \sum_{s=1}^n x_{rs}^j$; $\lambda = [\lambda^1, \lambda^2, \dots, \lambda^m]^T$.
- p_i^j : Price per unit resource on computer i for user j .
- $p^j = [p_1^j, p_2^j, \dots, p_n^j]^T$.

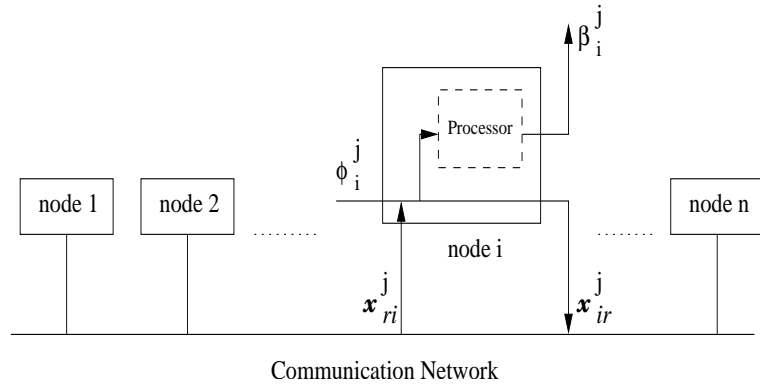


Figure 5.1: Grid System Model

A job arriving at node i may be either processed (locally) at node i or transferred to a neighboring node j through the communication network. We assume that a job can only be transferred once (*i.e.*, if it is transferred to node j , then it will be processed locally at node j). The decision of transferring a job does not depend on the state of the system and hence is *static* in nature. If a node i sends (receives) jobs to (from) node j , node j does not send (receive) jobs to (from) node i .

For each user j , nodes are classified into the following as in (Kim and Kameda 1990):

- Idle source (R_a^j): does not process any user j jobs ($\beta_i^j = 0$).
- Active source (R_a^j): processes some of the user j jobs that arrive and it sends the remaining user j jobs to other nodes. But, it does not receive any user j jobs from other nodes.
- Neutral (N^j): processes user j jobs locally without sending or receiving user j jobs.
- Sink (S^j): only receives user j jobs from other nodes but it does not send out any user j jobs.

Assuming that each computer is modeled as an M/M/1 queuing system (Kleinrock 1975), the expected node delay (queuing delay + processing delay) of an user j job processed at computer i is given by (the model is derived from queuing theory similar to Chapter 3):

$$F_i^j(\beta_i) = \frac{1}{(\mu_i - \sum_{k=1}^m \beta_i^k)} \quad (5.52)$$

We assume that the expected communication delay of a job from node r to node s is independent of the source-destination pair (r, s) but may depend on the total traffic through the network denoted by λ where $\lambda = \sum_{j=1}^m \lambda^j$. Modeling the communication network as an M/M/1 queuing system (Kleinrock 1975), the expected communication delay of an user j ($j = 1, \dots, m$) job is given by (the model is derived from queuing theory similar to Chapter 3):

$$G^j(\lambda) = \frac{t}{(1 - t \sum_{k=1}^m \lambda^k)}, \quad \sum_{k=1}^m \lambda^k < \frac{1}{t} \quad (5.53)$$

where t is the mean communication time for sending and receiving a job from one computer to the other for any user. Note that $F_i^j(\beta_i)$ and $G^j(\lambda)$ are increasing positive functions.

The network traffic of user j can be expressed in terms of the variable β_i^j as:

$$\lambda^j = \frac{1}{2} \sum_{i=1}^n |\phi_i^j - \beta_i^j| \quad (5.54)$$

Assuming that the nodes and the communication network have been expressed in the product-form of queuing network models (Jain 1991), the overall expected response time of user j can be written as the sum of the expected node delay and the expected communication delay as

follows (Kim and Kameda 1992):

$$T^j(\beta) = \frac{1}{\phi^j} \sum_{i=1}^n \beta_i^j F_i^j(\beta_i) + \frac{\lambda^j}{\phi^j} G^j(\lambda) \quad (5.55)$$

$$= \frac{1}{\phi^j} \sum_{i=1}^n \frac{\beta_i^j}{(\mu_i - \sum_{k=1}^m \beta_i^k)} + \frac{\lambda^j t}{\phi^j (1 - t \sum_{k=1}^m \lambda^k)} \quad (5.56)$$

Let k_i be a constant which maps the response time to the amount of resources consumed at computer i and let k_c be a constant which maps the communication delay to the amount of resources consumed from the communication network. We assume that the price the users have to pay for consuming a unit resource of the network is constant for all the users and denote it by p_c .

Thus, the overall expected cost of user j is given by:

$$D^j(\beta) = \frac{1}{\phi^j} \sum_{i=1}^n \frac{k_i p_c^j \beta_i^j}{(\mu_i - \sum_{k=1}^m \beta_i^k)} + \frac{k_c p_c t \lambda^j}{\phi^j (1 - t \sum_{k=1}^m \lambda^k)} \quad (5.57)$$

5.3 Non-cooperative Game among the Users

In this section, we formulate the job allocation problem as a non-cooperative game among the users. We use the game theory terminology introduced in (Grosu and Chronopoulos 2005). Each user j ($j = 1, \dots, m$) must find the workload (β_i^j) that is assigned to computer i such that the expected price of his own jobs ($D^j(\beta)$) is minimized. The vector $\beta^j = [\beta_1^j, \beta_2^j, \dots, \beta_n^j]$ is called the job allocation strategy of user j ($j = 1, \dots, m$) and the vector $\beta^* = [\beta^1, \beta^2, \dots, \beta^m]$ is called the strategy profile of the job allocation game. The strategy of user j depends on the job allocation strategies of the other users.

The assumptions for the existence of a feasible strategy profile are as follows:

- (i) *Positivity*: $\beta_i^j \geq 0$, $i = 1, \dots, n$, $j = 1, \dots, m$;
- (ii) *Conservation*: $\sum_{i=1}^n \beta_i^j = \phi^j$, $j = 1, \dots, m$;
- (iii) *Stability*: $\sum_{j=1}^m \beta_i^j < \mu_i$, $i = 1, \dots, n$;

A *Non-cooperative job allocation game* consists of a set of players, a set of strategies, and preferences over the set of strategy profiles:

- (i) *Players*: The m users.
- (ii) *Strategies*: Each user's set of feasible job allocation strategies.
- (iii) *Preferences*: Each user's preferences are represented by its expected price (D^j). Each user j prefers the strategy profile β^* to the strategy profile $\beta^{*'}$ if and only if $D^j(\beta^*) < D^j(\beta^{*'})$.

We need to solve the above game for our job allocation scheme. A solution can be obtained at the Nash equilibrium (Fudenberg and Tirole 1994) which is defined as follows.

Definition 5.3.1. (Nash equilibrium): A *Nash equilibrium* of the job allocation game defined above is a strategy profile β^* such that for every user j ($j = 1, \dots, m$):

$$\beta^j \in \arg \min_{\tilde{\beta}^j} D^j(\beta^1, \dots, \tilde{\beta}^j, \dots, \beta^m) \quad (5.58)$$

At the Nash equilibrium, a user j cannot further decrease its expected price by choosing a different job allocation strategy when the other users strategies are fixed. The equilibrium strategy profile can be found when each user's job allocation strategy is a *best response* to the other users strategies.

The *best response* for user j , is a solution to the following optimization problem (BR^j):

$$\min_{\beta^j} D^j(\beta) \quad (5.59)$$

subject to the constraints:

$$\beta_i^j \geq 0, \quad i = 1, \dots, n \quad (5.60)$$

$$\sum_{i=1}^n \beta_i^j = \phi^j \quad (5.61)$$

$$\sum_{j=1}^m \beta_i^j < \mu_i, \quad i = 1, \dots, n \quad (5.62)$$

Remark 5.3.1. In finding the solution to BR^j , the strategies of all the other users are kept fixed and so the variables in BR^j are the workloads of user j , *i.e.*, $\beta^j = (\beta_1^j, \beta_2^j, \dots, \beta_n^j)$.

In order to solve the optimization problem in eq. (5.59), we define the following functions:

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} [k_i p_i^j \beta_i^j F_i^j(\beta_i)] = \frac{k_i p_i^j \mu_i^j}{(\mu_i^j - \beta_i^j)^2} \quad (5.63)$$

where $\mu_i^j = \mu_i - \sum_{k=1, k \neq j}^m \beta_i^k$.

$$g^j(\lambda) = \frac{\partial}{\partial \lambda^j} [k_c p_c \lambda^j G^j(\lambda)] = \frac{k_c p_c t g_{-j}}{(g_{-j} - t \lambda^j)^2} \quad (5.64)$$

where $g_{-j} = (1 - t \sum_{k=1, k \neq j}^m \lambda^k)$.

$$(f_i^j)^{-1}(\beta_i |_{\beta_i^j = x}) = \begin{cases} (\mu_i^j - \sqrt{\frac{k_i p_i^j \mu_i^j}{x}}), & \text{if } x > \frac{k_i p_i^j}{\mu_i^j} \\ 0, & \text{if } x \leq \frac{k_i p_i^j}{\mu_i^j} \end{cases} \quad (5.65)$$

The *best response* strategy of user j , which is the solution of BR^j , is given in the following theorem.

Theorem 5.3.1. The solution to the optimization problem BR^j satisfies the relations

$$\begin{aligned} f_i^j(\beta_i) &\geq \alpha^j + g^j(\lambda), & \beta_i^j &= 0 & (i \in R_d^j), \\ f_i^j(\beta_i) &= \alpha^j + g^j(\lambda), & 0 < \beta_i^j &< \phi_i^j & (i \in R_a^j), \\ \alpha^j + g^j(\lambda) &\geq f_i^j(\beta_i) \geq \alpha^j, & \beta_i^j &= \phi_i^j & (i \in N^j), \\ f_i^j(\beta_i) &= \alpha^j, & \beta_i^j &> \phi_i^j & (i \in S^j), \end{aligned} \quad (5.66)$$

subject to the total flow constraint,

$$\sum_{i \in S^j} (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j}) + \sum_{i \in N^j} \phi_i^j + \sum_{i \in R_a^j} (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j + g^j(\lambda)}) = \phi^j \quad (5.67)$$

where α^j is the Lagrange multiplier.

Proof: In Appendix C. □

Since it is not possible to obtain a closed form solution for α^j from eq. (5.67), we use a binary search to solve eq. (5.67) iteratively for α^j similar to (Kim and Kameda 1990). Also, the following properties which are true in the optimal solution can be derived from Theorem 5.3.1 and

their proofs are similar to those in (Kim and Kameda 1990).

Property 5.3.1.

$$\begin{aligned}
f_i^j(\beta_i|\beta_i^j=0) &\geq \alpha^j + g^j(\lambda), \quad \text{iff } \beta_i^j = 0, \\
f_i^j(\beta_i|\beta_i^j=\phi_i^j) &> \alpha^j + g^j(\lambda) > f_i^j(\beta_i|\beta_i^j=0), \quad \text{iff } 0 < \beta_i^j < \phi_i^j, \\
\alpha^j \leq f_i^j(\beta_i|\beta_i^j=\phi_i^j) &\leq \alpha^j + g^j(\lambda), \quad \text{iff } \beta_i^j = \phi_i^j, \\
\alpha^j > f_i^j(\beta_i|\beta_i^j=\phi_i^j), &\quad \text{iff } \beta_i^j > \phi_i^j.
\end{aligned}$$

Property 5.3.2. If β^j is an optimal solution to the problem in eq. (5.59), then we have

$$\begin{aligned}
\beta_i^j &= 0, \quad i \in R_d^j, \\
\beta_i^j &= (f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j+g^j(\lambda)), \quad i \in R_a^j, \\
\beta_i^j &= \phi_i^j, \quad i \in N^j, \\
\beta_i^j &= (f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j), \quad i \in S^j.
\end{aligned}$$

Property 5.3.3. If β^j is an optimal solution to the problem in eq. (5.59), then we have $\lambda^j = \lambda_S^j = \lambda_R^j$,

where

$$\begin{aligned}
\lambda_S^j &= \sum_{i \in S^j} [(f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j) - \phi_i^j], \\
\lambda_R^j &= \sum_{i \in R_d^j} \phi_i^j + \sum_{i \in R_a^j} [\phi_i^j - (f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j+g^j(\lambda_S^j))].
\end{aligned}$$

Remark 5.3.2. The conditions in Property 5.3.1 help to partition the nodes into one of the four categories mentioned above for user j . Once the node partition for user j is known, his optimal loads can be calculated based on Property 5.3.2 Property 5.3.3 states that the job flow out of the sources equals the job flow into the sinks for each user j .

Based on the above properties, we can have the following definitions (eqs. (5.68) - (5.71)) for an arbitrary $\alpha^j (\geq 0)$.

$$S^j(\alpha^j) = \{i | f_i^j(\beta_i|\beta_i^j=\phi_i^j) < \alpha^j\} \quad (5.68)$$

$$\lambda_S^j(\alpha^j) = \sum_{i \in S^j(\alpha^j)} [(f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j) - \phi_i^j] \quad (5.69)$$

$$R_d^j(\alpha^j) = \{i | f_i^j(\beta_i|\beta_i^j=0) \geq \alpha^j + g^j(\lambda|\lambda^j=\lambda_S^j(\alpha^j))\} \quad (5.70)$$

$$R_a^j(\alpha^j) = \{i | f_i^j(\beta_i|\beta_i^j=\phi_i^j) > \alpha^j + g^j(\lambda|\lambda^j=\lambda_S^j(\alpha^j)) > f_i^j(\beta_i|\beta_i^j=0)\}$$

$$\lambda_R^j(\alpha^j) = \sum_{i \in R_d^j(\alpha^j)} \phi_i^j + \sum_{i \in R_a^j(\alpha^j)} [\phi_i^j - (f_i^j)^{-1}(\beta_i|\beta_i^j=\alpha^j+g^j(\lambda|\lambda^j=\lambda_S^j(\alpha^j)))]$$

$$N^j(\alpha^j) = \{i | \alpha^j \leq f_i^j(\beta_i |_{\beta_i^j = \phi_i^j}) \leq \alpha^j + g^j(\lambda |_{\lambda^j = \lambda_S^j(\alpha^j)})\} \quad (5.71)$$

Thus, if an optimal α^j is given, the node partitions in the optimal solution are characterized as

$$\begin{aligned} R_d^j &= R_d^j(\alpha^j), R_a^j = R_a^j(\alpha^j), N^j = N^j(\alpha^j), \\ S^j &= S^j(\alpha^j) \end{aligned} \quad (5.72)$$

and

$$\lambda^j = \lambda_S^j = \lambda_R^j = \lambda_S^j(\alpha^j) = \lambda_R^j(\alpha^j) \quad (5.73)$$

In the following, we present an algorithm for determining user j 's best response strategy:

BEST-RESPONSE algorithm:

Input: $\phi^j, \beta, \lambda, p^j, \mu_1, \dots, \mu_n, k_1, \dots, k_n$.

Output: β^j .

1. *Initialization:*

$$\beta_i^j \leftarrow \phi_i^j; i \in N^j; i = 1, \dots, n$$

2. Sort the computers such that

$$f_1^j(\beta_1 |_{\beta_1^j = \phi_1^j}) \leq \dots \leq f_n^j(\beta_n |_{\beta_n^j = \phi_n^j})$$

If $f_1^j(\beta_1 |_{\beta_1^j = \phi_1^j}) + g^j(\lambda |_{\lambda^j = 0}) \geq f_n^j(\beta_n |_{\beta_n^j = \phi_n^j})$,

STOP (No load balancing is required).

3. Determine α^j (using a binary search):

$$a \leftarrow f_1^j(\beta_1 |_{\beta_1^j = \phi_1^j})$$

$$b \leftarrow f_n^j(\beta_n |_{\beta_n^j = \phi_n^j})$$

while(1) **do**

$$\lambda_S^j(\alpha^j) \leftarrow 0$$

$$\lambda_R^j(\alpha^j) \leftarrow 0$$

$$\alpha^j \leftarrow \frac{a+b}{2}$$

Calculate: $S^j(\alpha^j), \lambda_S^j(\alpha^j), R_d^j(\alpha^j),$

$R_a^j(\alpha^j)$, and $\lambda_R^j(\alpha^j)$ (eqs. (5.68) - (5.71))

in the order given for $i = 1, \dots, n$

If $(|\lambda_S^j(\alpha^j) - \lambda_R^j(\alpha^j)| < \epsilon)$ **EXIT**

If $(\lambda_S^j(\alpha^j) > \lambda_R^j(\alpha^j))$

$b \leftarrow \alpha^j$

else

$a \leftarrow \alpha^j$

4. Determine user j 's loads on the computers:

$\beta_i^j \leftarrow 0$, for $i \in R_d^j(\alpha^j)$

$\beta_i^j \leftarrow (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j + g^j(\lambda)})$, for $i \in R_a^j(\alpha^j)$

$\beta_i^j \leftarrow (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j})$, for $i \in S^j(\alpha^j)$

$\beta_i^j \leftarrow \phi_i^j$, for $i \in N^j(\alpha^j)$ (eq. (5.71))

Remark 5.3.3. The running time of this algorithm is $O(n \log n + n \log 1/\epsilon)$, where ϵ denotes the tolerance used for computing α^j in step 3 of the algorithm.

In order to obtain the equilibrium allocation, we need an iterative algorithm where each user updates his strategies (by computing his *best response*) periodically by fixing the other users strategies. We can set a virtual ring topology of the users to communicate and iteratively apply the BEST-RESPONSE algorithm to compute the Nash equilibrium similar to (Grosu and Chronopoulos 2005).

In the following, we present the iterative algorithm (NASHPC) for computing the Nash equilibrium for our non-cooperative job allocation game. One of the users can initiate the algorithm (initiating user) who calculates his initial strategies by fixing the other users strategies to zero. An iteration is said to be complete if this *initiating user* receives a message from his left neighbor. He then checks if the error norm is less than a tolerance in which case he sends a terminating message to his right neighbor to be propagated around the ring.

NASHPC distributed job allocation algorithm:

Each user j , $j = 1, \dots, m$ in the ring performs the following steps in each iteration:

1. Receive the current strategies of all the other users from the left neighbor.
2. If the message is a termination message, then pass the termination message to the right neighbor

and EXIT.

3. Update the strategies (β^j) by calling the BEST-RESPONSE
4. Calculate D^j (eq. (5.57)) and update the error norm.
5. Send the updated strategies and the error norm to the right neighbor.

This algorithm can be restarted periodically or when the system parameters are changed. Users will use the strategies that are computed (at the Nash equilibrium) and the system remains in equilibrium.

5.4 Modeling Results

We run a computer model to study the impact of system utilization and heterogeneity on the performance of the NASHPC scheme. We also implemented the following job allocation scheme for comparison purposes:

- *Global Optimal Scheme with Pricing and Communication (GOSPC)* : This scheme minimizes the expected cost over all the jobs executed by the grid cluster. This is an extension of the *overall optimal scheme* (Kim and Kameda 1990) in the multi-class environment to include pricing. The loads (β_i^j) for each user are obtained by solving the following non-linear optimization problem:

$$\min D(\beta) = \frac{1}{\Phi} \sum_{j=1}^m \left[\sum_{i=1}^n k_i p_i^j \beta_i^j F_i^j(\beta_i) + k_c p_c \lambda^j G^j(\lambda) \right] \quad (5.74)$$

subject to the constraints:

$$\sum_{i=1}^n \beta_i^j = \phi^j \quad j = 1, \dots, m \quad (5.75)$$

$$\beta_i^j \geq 0, \quad i = 1, \dots, n; j = 1, \dots, m \quad (5.76)$$

The performance metrics used in our model are the *expected response time* and the *fairness index* (Jain 1991). The *fairness index* (defined from the users' perspective),

$$I(\mathbf{C}) = \frac{[\sum_{j=1}^m C_j]^2}{m \sum_{j=1}^m C_j^2} \quad (5.77)$$

Table 5.1: System configuration

Relative service rate	1	2	5	10
Number of computers	6	5	3	2
Service rate (jobs/sec)	10	20	50	100
k_i	1	2	3	4

is used to quantify the fairness of job allocation schemes. The parameter \mathbf{C} is the vector $\mathbf{C} = (C_1, C_2, \dots, C_m)$ where C_j is the expected cost of user j 's jobs. If all the users have the same total expected price, then $I = 1$ and the system is 100% fair to all users and it is cost-balanced. If I decreases, then the job allocation scheme favors only some users. In the following we present and discuss the modeling results.

5.4.1 Effect of System Utilization

System utilization (ρ) represents the amount of load on the system. It is defined as the ratio of the total arrival rate to the aggregate service rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^n \mu_i} \quad (5.78)$$

We modeled a heterogeneous system consisting of 16 computers to study the effect of system utilization. The system has computers with four different service rates and is shared by 10 users. The system configuration is shown in Table 5.1. The first row contains the relative service rates of each of the four computer types. The relative service rate for computer C_i is defined as the ratio of the service rate of C_i to the service 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 third row shows the service rate of each computer type in the system. The last row shows the values for k_i . It is assigned based on the service rate of the computer (Ghosh, Roy, Basu, and Das 2004). The price vector p^j for each user is obtained based on the alternating offer bargaining game described in section I. k_c and p_c are assumed to be 1 in all the following model demonstrations.

The total job arrival rate in the system Φ is determined by the system utilization ρ and the aggregate service rate of the system. The total job arrival rate Φ is chosen by fixing the system

utilization. The job arrival rate for each user $\phi^j, j = 1, \dots, 10$ is determined from the total arrival rate as $\phi^j = q^j \Phi$, where the fractions q^j are given in Table 5.2. The job arrival rates of each user $j, j = 1, \dots, 10$ to each computer $i, i = 1, \dots, 16, i.e.,$ the ϕ_i^j 's are obtained in a similar manner. The mean communication time t is assumed to be 0.001 sec.

Table 5.2: Job arrival fractions q^j for each user

User	1	2	3-6	7	8-10
q^j	0.3	0.2	0.1	0.07	0.01

In Figure 5.2, 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 of the NASHPC scheme which minimizes the cost of each user is very close to that of GOSPC which minimizes the cost of the entire system.

Figure 5.3 shows the fairness index for different values of system utilization. The fairness index of GOSPC varies from 1 at low load, to 0.95 at high load. The NASHPC scheme has a fairness index close to 1 and each user obtains the minimum possible expected price for its own jobs (*i.e.*, it is user-optimal). So, GOSPC scheme whose objective is to reduce the overall cost of the grid cluster is unfair whereas NASHPC is fair to each user.

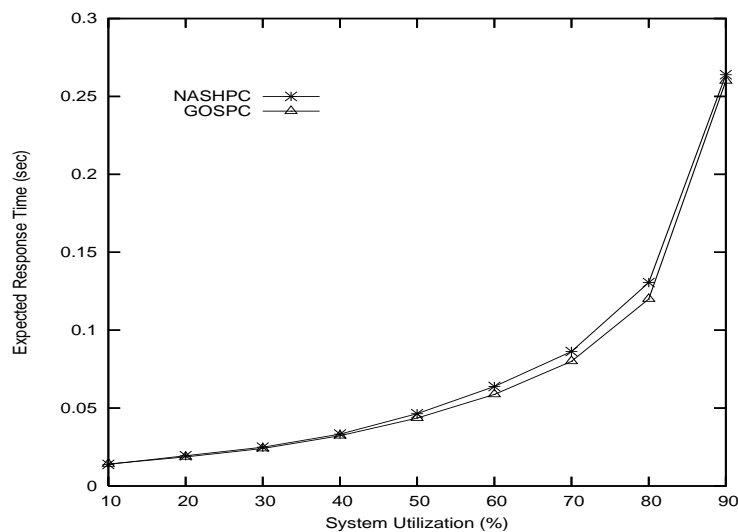


Figure 5.2: System Utilization vs Expected Response Time

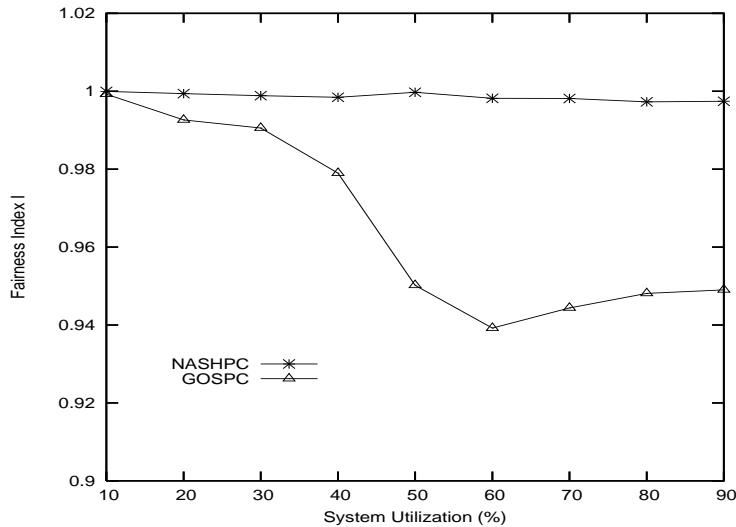


Figure 5.3: System Utilization vs Fairness Index

5.4.2 Effect of Heterogeneity

Here we study the effect of heterogeneity on the performance of our job allocation scheme. One of the common measures of heterogeneity is the *speed skewness* (Tang and Chanson 2000) which is defined as the ratio of maximum service rate to minimum service rate of the grid computers. The effectiveness of NASHPC is studied by varying the speed skewness.

We modeled a heterogeneous system consisting of 16 computers (2 fast and 14 slow) to study the effect of heterogeneity. The slow computers have a relative service rate of 1 and the relative service rate of the fast computers is varied from 1 (homogeneous system) to 20 (highly heterogeneous system). The system utilization was kept constant ($\rho = 60\%$) and the mean communication time t is assumed to be 0.001 sec. In Table 5.3, we present the service rates (μ_i jobs/sec) of the computers in the systems and the total arrival rates (Φ) for some of the cases. C1 and C2 represent the fast computers and C3 through C16 represent the slow computers. The fractions used to determine the job arrival rate of each user are those presented in Table 5.2.

Table 5.3: System parameters

Speed skewness	1	4	8	12	16	20
μ_i of C1,C2	10	40	80	120	160	200
μ_i of C3-C16	10	10	10	10	10	10
Φ (jobs/sec)	96	132	180	228	276	324

Figure 5.4 shows the effect of speed skewness on the expected response time of the two

schemes. It can be observed that, increasing the speed skewness, the NASHPC and GOSPC schemes yield almost the same expected response time which means that in highly heterogeneous grid systems the NASHPC scheme is very effective.

From Figure 5.5, it can be observed that the fairness index of NASHPC is very close to 1. The fairness index of GOSPC drops from 1 at low skewness to 0.46 at high skewness. This shows that the GOSPC scheme produces an allocation which does not guarantee equal expected price for all the users in the grid cluster. The performance of NASHPC scheme is close to that of GOSPC with the additional advantage of user-optimality.

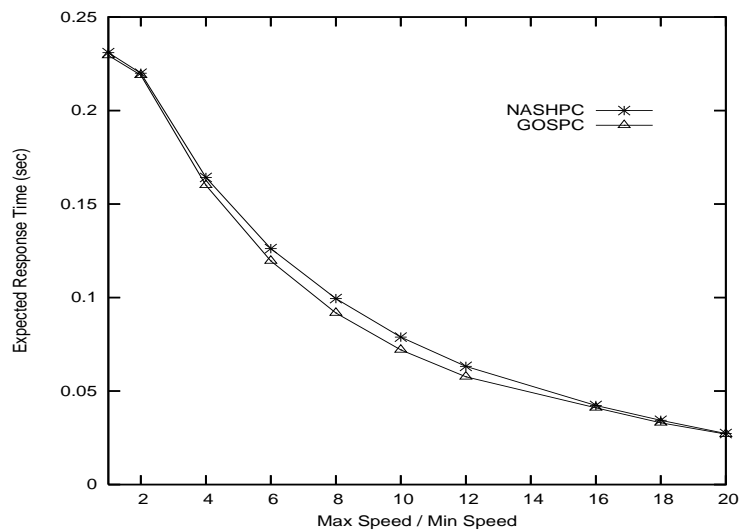


Figure 5.4: Heterogeneity vs Expected Response Time

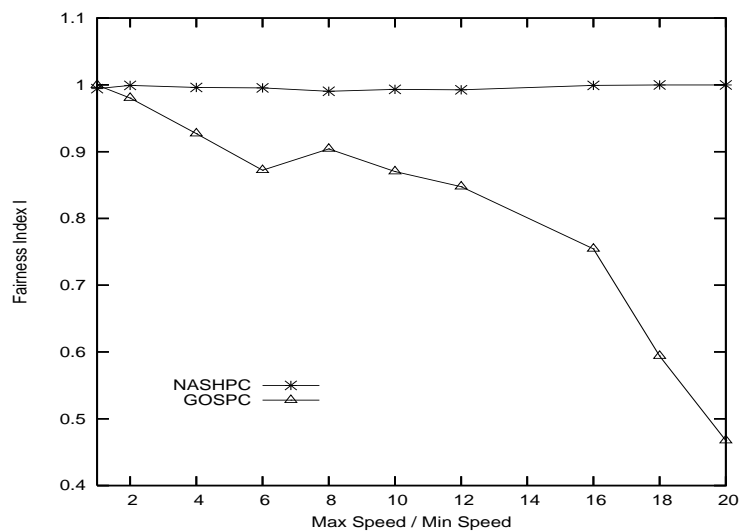


Figure 5.5: Heterogeneity vs Fairness Index

5.5 Summary

In this chapter, we proposed a price-based user-optimal job allocation scheme for grid systems. The nodes in the system are connected by a communication network. The job allocation problem is formulated as a non-cooperative game among the users whose objective is to minimize the expected cost of their own jobs. We used the Nash equilibrium as the solution of our game and proposed an algorithm to compute the load allocation of the users at the equilibrium. We ran a grid model with various system loads and configurations and compared our scheme (NASHPC) with an overall optimal scheme (GOSPC). Based on the model results, we observed that the performance of the NASHPC scheme is not only comparable with that of GOSPC in terms of the expected response time, but also provides an allocation which is fair (in terms of cost) to all the grid users.

In the next chapter, we extend the two static schemes, NASHPC and GOSPC, to dynamic load balancing schemes for multi-user job distributed systems.

CHAPTER 6: Dynamic Load Balancing in Distributed Systems

6.1 Introduction

Load balancing is one of the most important problems in attaining high performance in distributed systems which may consist of many heterogeneous resources connected via one or more communication networks. In these distributed systems it is possible for some computers to experience heavy load while others experience light load. This situation can lead to poor overall system performance. The goal of load balancing is to improve the performance of the system by balancing the load among the computers.

The load balancing schemes can be either *static* or *dynamic* (Shivaratri, Krueger, and Singhal 1992). The static schemes either do not use any system information or use only the average system behavior whereas the dynamic schemes consider instantaneous system states in the job allocation calculations. Also, jobs in a distributed system can be divided into different classes (multi-class or multi-user) based on their resource usage characteristics and ownership.

6.1.1 Related Work

There are many studies on static load balancing for single-user (single-class) and multi-user jobs that provide system-optimal solution ((Kameda, Li, Kim, and Zhang 1997; Tang and Chanson 2000; Tantawi and Towsley 1985; Li and Kameda 1994; Lee 1995) and references therein). These schemes determine a load allocation to obtain a system-wide optimal expected response time but the *fairness* of allocation to the individual jobs or users is not considered *i.e.*, some jobs or users may experience much longer expected response time than others.

Few studies exist on static load balancing that provide individual-optimal and user-optimal solutions which are based on game theory. Individual and user-optimal policies for an infinite number of jobs or users based on *non-cooperative* games using *Wardrop equilibrium* are studied in (Kameda, Li, Kim, and Zhang 1997). Individual-optimal policies for finite jobs based on *cooperative* game theory are studied in (Grosu, Chronopoulos, and Leung 2002; Penmatsa and Chronopoulos 2006a). User-optimal policies based on *Nash equilibrium* are studied in (Grosu and Chronopoulos 2005; Penmatsa and Chronopoulos 2005; Penmatsa and Chronopoulos 2006b) for finite number of users. Non-

cooperative game theory was also used to model grid systems (Kwok, Hwang, and Song 2007; Ghosh, Roy, Das, and Basu 2005) and for price-based job allocation in distributed systems (Ghosh, Basu, and Das 2007).

Many dynamic load balancing policies exist ((Shivaratri, Krueger, and Singhal 1992; El-Zoghdy, Kameda, and Li 2002; Anand, Ghose, and Mani 1999; Zeng and Veeravalli 2006; Zeng and Bharadwaj 2004; Cai, Lee, Heng, and Zhu 1997; Mitzenmacher 1997; Hui and Chanson 1999; Kulkarni and Sengupta 2000; Han, Shin, and Yun 2000) and references therein). Kameda *et al.* (Zhang, Kameda, and Hung 1997) studied dynamic load balancing for single-class jobs to provide system-optimal and individual-optimal solutions. Their dynamic scheme, which tries to provide an individual-optimum solution, is based on a static individual-optimum scheme (Kameda, Li, Kim, and Zhang 1997), which uses the *Wardrop equilibrium* as the solution.

6.1.2 Our Contribution

Most of the above dynamic policies are for single-class jobs and their main objective is to minimize the expected response time of the entire system. However, in this case some individual user jobs experience much longer expected response time than other user jobs. This may not be acceptable in current distributed systems, where the users have requirements for fast job execution. In this chapter (Penmatsa and Chronopoulos 2007) we propose two dynamic load balancing schemes for multi-user jobs in heterogeneous distributed systems. The computers in the distributed system are connected by a communication network. We review two existing static load balancing schemes and extend them to dynamic schemes. These two existing load balancing schemes differ in their objective. (i) The first scheme, GOS (Kameda, Li, Kim, and Zhang 1997) tries to minimize the expected response time of the entire system. (ii) The second scheme, NCOOPC tries to minimize the expected response time of the individual users (to provide a user-optimal solution). We base our dynamic schemes, DGOS and DNCOOPC, on the static schemes (i) and (ii) respectively. DGOS tries to dynamically balance the load among the computers to obtain a system-optimal solution but may not be fair to the users. DNCOOPC tries to dynamically balance the load among the computers to obtain a user-optimal solution. This solution provides fairness to all the users so that they all have approximately the same expected response time independent of the computers allocated for their jobs. The performance of the

dynamic schemes is compared with that of the static schemes by running a computer model under various system conditions. The results show that, at low communication overheads, the dynamic schemes show superior performance over the static schemes. But as the overheads increase, the dynamic schemes tend to perform similar to that of the static schemes.

6.1.3 Chapter Organization

The rest of the chapter is organized as follows. In Section 6.2, we present the system model. In Section 6.3, we review the two static load balancing schemes based on which the dynamic schemes are proposed in Section 6.4. The performance of the dynamic schemes is compared with that of the static schemes using computer modeling in Section 6.5. A summary of the chapter is provided in Section 6.6.

6.2 Distributed System Model

We consider a distributed system model as shown in Figure 6.1. The system has ‘ n ’ nodes connected by a communication network. The nodes and the communication network are modeled as M/M/1 queuing systems (*i.e.*, Poisson arrivals and exponentially distributed processing times) (Jain 1991). Jobs arriving at each node may belong to ‘ m ’ different users (classes).

The terminology, notations and assumptions used are as follows:

- a_i^j : Mean inter-arrival time of a user j job to node i .
- ϕ_i^j : Mean job arrival rate of user j to node i ($\phi_i^j = \frac{1}{a_i^j}$).
- ϕ^j : Total job arrival rate of user j .
- $\phi^j = \sum_{i=1}^n \phi_i^j$
- Φ : Total job arrival rate of the system.
- $\Phi = \sum_{j=1}^m \phi^j$
- r_i : Mean service time of a job at node i .
- μ_i : Mean service rate of node i ($\mu_i = \frac{1}{r_i}$).

- β_i^j : Job processing rate (load) of user j at node i .
- $\beta_i = [\beta_i^1, \beta_i^2, \dots, \beta_i^m]^T$, is the vector of loads at node i from users $1, \dots, m$.
- $\beta = [\beta_1, \beta_2, \dots, \beta_n]^T$, is the load vector of all nodes $i = 1, \dots, n$ (from all users $1, \dots, m$).
- $\beta^k = [\beta_1^k, \beta_2^k, \dots, \beta_n^k]^T$, is the vector of loads of user k to nodes $1, \dots, n$.
- x_{rs}^j : Job flow rate of user j from node r to node s .
- λ^j : Job traffic through the network of user j .
- $\lambda^j = \sum_{r=1}^n \sum_{s=1}^n x_{rs}^j$
- $\lambda = [\lambda^1, \lambda^2, \dots, \lambda^m]^T$; $\lambda = \sum_{j=1}^m \lambda^j$.
- N_i^j : Mean number of jobs of user j at node i .
- n_i^j : Number of jobs of user j at node i at a given instant.
- P : Time period for the exchange of job count of each user by all the nodes.
- $P1$: Time period for the exchange of arrival rates of each user by all the nodes.
- t : Mean communication time for sending or receiving a job from one node to another for any user.
- ρ : Mean utilization of the communication network ($\rho = t \sum_{k=1}^m \lambda^k$).
- ρ^{-j} : Mean utilization of the communication network excluding user j traffic ($\rho^{-j} = t \sum_{k=1, k \neq j}^m \lambda^k$).
- Jobs arriving from various users to a node differ in their arrival rates but have the same processing time (*i.e.*, all the jobs are assumed to be of the same size).

6.3 Static Load Balancing

In this section, we study two static load balancing schemes based on which two dynamic load balancing schemes are derived in the next section. These static schemes are also used for comparison in Section 6.5 to evaluate the performance of the dynamic schemes.

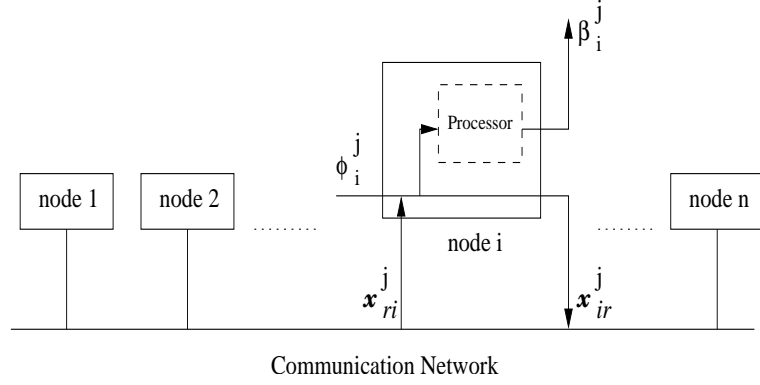


Figure 6.1: Distributed System Model

Following are the assumptions for static load balancing, similar to (Kameda, Li, Kim, and Zhang 1997). A job arriving at node c may be either processed at node c or transferred to another node d for remote processing through the communication network. A job can only be transferred at most once. The decision of transferring a job does not depend on the state of the system and hence is *static* in nature. If a node c sends (receives) jobs to (from) node d , node d does not send (receive) jobs to (from) node c .

For each user j , nodes are classified into the following as in (Kameda, Li, Kim, and Zhang 1997):

- Idle source (R_a^j): does not process any user j jobs ($\beta_i^j = 0$).
- Active source (R_a^j): processes some of the user j jobs that arrive and it sends the remaining user j jobs to other nodes. But, it does not receive any user j jobs from other nodes.
- Neutral (N^j): processes user j jobs locally without sending or receiving user j jobs.
- Sink (S^j): only receives user j jobs from other nodes but it does not send out any user j jobs.

The expected response time of a user j job processed at node i (queuing delay + processing time) is denoted by $F_i^j(\beta_i)$. The expected communication delay of a user j job from node c to node d is denoted by $G^j(\lambda)$ and it is assumed to be independent of the source-destination pair (c, d) but it may depend on the total traffic through the network, λ .

Based on our assumptions on the node and the network models in the previous section, we have the following relations for the node and communication delays (Jain 1991) (the model is derived

from queuing theory similar to Chapter 3):

$$F_i^j(\beta_i) = \frac{1}{(\mu_i - \sum_{k=1}^m \beta_i^k)} \quad (6.79)$$

$$G^j(\lambda) = \frac{t}{(1 - t \sum_{k=1}^m \lambda^k)}, \quad \sum_{k=1}^m \lambda^k < \frac{1}{t} \quad (6.80)$$

6.3.1 Global Optimal Scheme (GOS)

This static load balancing scheme is proposed and studied in (Kameda, Li, Kim, and Zhang 1997). It minimizes the expected job response time of the entire system. The problem of minimizing the entire system expected job response time is expressed as (the nodes and the communication network are assumed to have the product-form of queuing network models (Jain 1991))

$$\min D(\beta) = \frac{1}{\Phi} \sum_{j=1}^m [\lambda^j G^j(\lambda) + \sum_{i=1}^n \beta_i^j F_i^j(\beta_i)] \quad (6.81)$$

subject to the constraints:

$$\sum_{i=1}^n \beta_i^j = \phi^j \quad j = 1, \dots, m \quad (6.82)$$

$$\beta_i^j \geq 0, \quad i = 1, \dots, n; j = 1, \dots, m \quad (6.83)$$

The above nonlinear optimization problem is solved by using the Kuhn-Tucker theorem and an algorithm to compute the optimal loads (β_i^j) is presented in (Kameda, Li, Kim, and Zhang 1997).

The user j marginal node delay $f_i^j(\beta_i)$ and marginal communication delay $g^j(\lambda)$ are defined as

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} \sum_{k=1}^m \beta_i^k F_i^k(\beta_i) = \frac{\mu_i}{(\mu_i - \sum_{k=1}^m \beta_i^k)^2} \quad (6.84)$$

$$g^j(\lambda) = \frac{\partial}{\partial \lambda^j} \sum_{k=1}^m \lambda^k G^k(\lambda) = \frac{t}{(1 - t \sum_{k=1}^m \lambda^k)^2} \quad (6.85)$$

where $\beta_i^k F_i^k(\beta_i)$ denotes the mean number of user k jobs in node i and $\lambda^k G^k(\lambda)$ denotes the mean number of user k jobs in the communication network.

6.3.2 Non-cooperative Scheme (NCOOPC)

Here, we review a static load balancing scheme whose goal is to minimize the expected job response time of the individual users (*i.e.*, to obtain a user-optimal solution). This problem is formulated, taking into account the users mean node delays (queuing and processing delays) and the mean communication delays, as a non-cooperative game among the users.

This non-cooperative approach for multi-user job allocation with communication and pricing (NASHPC) for grid systems was studied in Chapter 5. The grid users try to minimize the expected cost of their own jobs. The problem was formulated as a non-cooperative game among the grid users and the Nash equilibrium provides the solution. Here, we are interested in the load balancing algorithm for multi-user jobs without pricing. Therefore, we take $k_i = 1$ ($i = 1, \dots, n$); $p_i^j = 1$ ($i = 1, \dots, n, j = 1, \dots, m$); and $k_c = p_c = 1$ in the following objective function of Chapter 5 which denotes the overall expected cost of user j :

$$Obj^j(\beta) = \frac{k_c p_c t \lambda^j}{\phi^j (1 - t \sum_{k=1}^m \lambda^k)} + \frac{1}{\phi^j} \sum_{i=1}^n \frac{k_i p_i^j \beta_i^j}{(\mu_i - \sum_{k=1}^m \beta_i^k)} \quad (6.86)$$

Then, we obtain a non-cooperative load balancing game where the users try to minimize the expected response time of their own jobs.

Each user j ($j = 1, \dots, m$) must find his workload (β_i^j) that is assigned to node i ($i = 1, \dots, n$) such that the expected response time of his own jobs ($D^j(\beta)$) is minimized, where

$$D^j(\beta) = \frac{1}{\phi^j} [\lambda^j G^j(\lambda) + \sum_{i=1}^n \beta_i^j F_i^j(\beta_i)] = \frac{1}{\phi^j} \left[\frac{\lambda^j t}{(1 - t \sum_{k=1}^m \lambda^k)} + \sum_{i=1}^n \frac{\beta_i^j}{(\mu_i - \sum_{k=1}^m \beta_i^k)} \right] \quad (6.87)$$

Each user minimizes his own expected response time independently of the others and they all eventually reach an equilibrium. We use the concept of Nash equilibrium (Fudenberg and Tirole 1994) as the solution of this non-cooperative game. At the equilibrium, a user cannot decrease his own expected response time any further by changing his decision alone.

The vector $\beta^j = [\beta_1^j, \beta_2^j, \dots, \beta_n^j]$ is called the load balancing strategy of user j ($j = 1, \dots, m$) and the vector $\beta^* = [\beta^1, \beta^2, \dots, \beta^m]$ is called the strategy profile of the load balancing game. The strategy of user j depends on the load balancing strategies of the other users.

At the Nash equilibrium, a user j cannot further decrease his expected response time by

choosing a different load balancing strategy when the other users strategies are fixed. The equilibrium strategy profile can be found when each user's load balancing strategy is a *best response* to the other users strategies.

The *best response* for user j , is a solution to the following optimization problem:

$$\min_{\beta^j} D^j(\beta) \quad (6.88)$$

subject to the following constraints for the existence of a feasible strategy profile:

$$\beta_i^j \geq 0, \quad i = 1, \dots, n \quad (6.89)$$

$$\sum_{i=1}^n \beta_i^j = \phi^j \quad (6.90)$$

$$\sum_{j=1}^m \beta_i^j < \mu_i, \quad i = 1, \dots, n \quad (6.91)$$

We define the following functions for each user j :

$$\hat{f}_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} (\beta_i^j F_i^j(\beta_i)) = \frac{(\mu_i - \sum_{k=1, k \neq j}^m \beta_i^k)}{(\mu_i - \sum_{k=1}^m \beta_i^k)^2} \quad (6.92)$$

$$\hat{g}^j(\lambda) = \frac{\partial}{\partial \lambda^j} (\lambda^j G^j(\lambda)) = \frac{t(1 - t \sum_{k=1, k \neq j}^m \lambda^k)}{(1 - t \sum_{k=1}^m \lambda^k)^2} \quad (6.93)$$

$$(\hat{f}_i^j)^{-1}(\beta_i |_{\beta_i^j=x}) = \begin{cases} (\mu_i^j - \sqrt{\frac{\mu_i^j}{x}}), & \text{if } x > \frac{1}{\mu_i^j} \\ 0, & \text{if } x \leq \frac{1}{\mu_i^j} \end{cases} \quad (6.94)$$

The solution to the optimization problem in eq. (6.88) is given in the following theorem which is similar to Theorem 5.3.1 in Chapter 5 where the $f_i^j(\beta_i)$ and $g^j(\lambda)$ in Theorem 5.3.1 of Chapter 5 are replaced by the above eqs. $\hat{f}_i^j(\beta_i)$ and $\hat{g}^j(\lambda)$ (eqs. (6.92) and (6.93)).

Theorem 6.3.1. The solution to the optimization problem in eq. (6.88) satisfies the relations

$$\begin{aligned}
\hat{f}_i^j(\beta_i) &\geq \alpha^j + \hat{g}^j(\lambda), & \beta_i^j &= 0 & (i \in R_a^j), \\
\hat{f}_i^j(\beta_i) &= \alpha^j + \hat{g}^j(\lambda), & 0 < \beta_i^j &< \phi_i^j & (i \in R_a^j), \\
\alpha^j + \hat{g}^j(\lambda) &\geq \hat{f}_i^j(\beta_i) \geq \alpha^j, & \beta_i^j &= \phi_i^j & (i \in N^j), \\
\hat{f}_i^j(\beta_i) &= \alpha^j, & \beta_i^j &> \phi_i^j & (i \in S^j),
\end{aligned} \tag{6.95}$$

subject to the total flow constraint,

$$\sum_{i \in S^j} (\hat{f}_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j}) + \sum_{i \in N^j} \phi_i^j + \sum_{i \in R_a^j} (\hat{f}_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j + \hat{g}^j(\lambda)}) = \phi^j \tag{6.96}$$

where α^j is the Lagrange multiplier.

Proof: Similar to Theorem 5.3.1 in Chapter 5. □

The α^j is used as the Lagrange multiplier which categorizes the nodes for each user j into sets of Active source nodes ($R_a^j(\alpha^j)$), Idle source nodes ($R_d^j(\alpha^j)$), Neutral nodes ($N^j(\alpha^j)$) and Sink nodes ($S^j(\alpha^j)$). The $\lambda_S^j(\alpha^j)$ denotes the network traffic of user j because of jobs sent by the set of Active and Idle source nodes as determined by α^j and $\lambda_R^j(\alpha^j)$ is the network traffic of user j because of jobs received by the set of Sink nodes as determined by α^j .

The *best response* of user j ($j = 1, \dots, m$) can be determined using the following algorithm:

BEST-RESPONSE algorithm:

Input: $\phi^j, \beta, \lambda, \mu_1, \dots, \mu_n$.

Output: β^j .

1. *Initialization:*

$$\beta_i^j \leftarrow \phi_i^j; i \in N^j; i = 1, \dots, n$$

2. Sort the computers such that

$$\hat{f}_1^j(\beta_1 |_{\beta_1^j = \phi_1^j}) \leq \dots \leq \hat{f}_n^j(\beta_n |_{\beta_n^j = \phi_n^j}).$$

If $\hat{f}_1^j(\beta_1 |_{\beta_1^j = \phi_1^j}) + \hat{g}^j(\lambda |_{\lambda^j = 0}) \geq \hat{f}_n^j(\beta_n |_{\beta_n^j = \phi_n^j})$, then no load balancing is required.

3. Determine the Lagrange multiplier α^j using a binary search:

$$a \leftarrow \hat{f}_1^j(\beta_1 |_{\beta_1^j = \phi_1^j})$$

$$b \leftarrow \hat{f}_n^j(\beta_n |_{\beta_n^j = \phi_n^j})$$

while(1) do

$$\lambda_S^j(\alpha^j) \leftarrow 0$$

$$\lambda_R^j(\alpha^j) \leftarrow 0$$

$$\alpha^j \leftarrow \frac{a+b}{2}$$

Calculate: $S^j(\alpha^j)$, $\lambda_S^j(\alpha^j)$, $R_d^j(\alpha^j)$, $R_a^j(\alpha^j)$, and $\lambda_R^j(\alpha^j)$

in the order given for $i = 1, \dots, n$.

If $(|\lambda_S^j(\alpha^j) - \lambda_R^j(\alpha^j)| < \epsilon)$ **EXIT**

If $(\lambda_S^j(\alpha^j) > \lambda_R^j(\alpha^j))$

$$b \leftarrow \alpha^j$$

else

$$a \leftarrow \alpha^j$$

4. Determine user j 's optimal loads:

$$\beta_i^j \leftarrow 0, \quad \text{for } i \in R_d^j(\alpha^j)$$

$$\beta_i^j \leftarrow (\hat{f}_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j + \hat{g}^j(\lambda)}), \quad \text{for } i \in R_a^j(\alpha^j)$$

$$\beta_i^j \leftarrow (\hat{f}_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j}), \quad \text{for } i \in S^j(\alpha^j)$$

$$\beta_i^j \leftarrow \phi_i^j, \quad \text{for } i \in N^j(\alpha^j)$$

The optimal loads of each user j ($j = 1, \dots, m$) *i.e.*, the equilibrium solution, can be obtained using an iterative algorithm where each user updates his strategies (by computing his *best response*) periodically by fixing the other users strategies. The users form a virtual ring topology to communicate and iteratively apply the BEST-RESPONSE algorithm to compute the Nash equilibrium. This is implemented in the following algorithm.

NCOOPC distributed load balancing algorithm:

Each user j , $j = 1, \dots, m$ in the ring performs the following steps in each iteration:

1. Receive the current strategies of all the other users from the left neighbor.
2. If the message is a termination message, then pass the termination message to the right neighbor and EXIT.
3. Update the strategies (β^j) by calling the BEST-RESPONSE.

4. Calculate D^j (eq. (6.87)) and update the error norm.
5. Send the updated strategies and the error norm to the right neighbor.

At the equilibrium solution, the $\hat{f}_i^j(\beta_i)$'s of each user at all the nodes are balanced.

6.4 Dynamic Load Balancing

Kameda *et al.* (Zhang, Kameda, and Hung 1997) proposed and studied dynamic load balancing algorithms for single-user job streams based on M/M/1 queues in heterogeneous distributed systems. We follow a similar approach to extend GOS and NCOOPC to dynamic schemes. In this section, we propose these two distributed dynamic load balancing schemes for multi-user jobs.

A distributed dynamic scheme has three components: 1) an *information policy* used to disseminate load information among nodes, 2) a *transfer policy* that determines whether job transfer activities are needed by a computer, and 3) a *location policy* that determines which nodes are suitable to participate in load exchanges.

The dynamic schemes which we propose use the number of jobs waiting in the queue to be processed (queue length) as the state information. The information policy is a periodic policy where the state information is exchanged between the nodes every P time units. When a job arrives at a node, the transfer policy component determines whether the job should be processed locally or should be transferred to another node for processing. If the job is eligible for transfer, the location policy component determines the destination node for remote processing.

In the following we discuss the dynamic load balancing schemes.

6.4.1 Dynamic Global Optimal Scheme (DGOS)

The goal of DGOS is to balance the workload among the nodes dynamically in order to obtain a system-wide optimization *i.e.*, to minimize the expected response time of all the jobs over the entire system. We base the derivation of DGOS on the static GOS of Section 3.

We use the following proposition to express the marginal node delay of a user j job at node i ($f_i^j(\beta_i)$ in eq. (6.84)) in terms of the mean service time at node i (r_i) and the mean number of jobs of user j at node i (N_i^j , $j = 1, \dots, m$).

Proposition 6.4.1. The user j marginal node delay in eq. (6.84) can be expressed in terms of r_i and N_i^j ($j = 1, \dots, m$) as

$$f_i^j(\beta_i) = r_i \left(1 + \sum_{k=1}^m N_i^k\right)^2 \quad (6.97)$$

Proof: In Appendix D. □

Rewriting eq. (6.85) in terms of ρ , we have

$$g^j(\lambda) = \frac{t}{(1 - \rho)^2}, \quad \rho < 1 \quad (6.98)$$

We next express $f_i^j()$ of Proposition 6.4.1 and $g^j()$ from eq. (6.98), which use the mean estimates of the system parameters, in terms of instantaneous variables.

Let ρ' denote the utilization of the network at a given instant. Replacing N_i^j 's, the mean number of jobs of users at node i by n_i^j 's and ρ , the mean utilization of the network by ρ' , in eqs. (6.97) and (6.98), we obtain (for user j)

$$f_i^j = r_i \left(1 + \sum_{k=1}^m n_i^k\right)^2 \quad (6.99)$$

$$g^j = \frac{t}{(1 - \rho')^2}, \quad \rho' < 1 \quad (6.100)$$

The above relations are used as the estimates of the user j *marginal virtual node delay* at node i and user j *marginal communication delay*. Whereas GOS tries to balance the marginal node delays of each user at all the nodes statically, DGOS will be derived to balance the marginal virtual node delays of each user at all the nodes dynamically. For a user u job arriving at node i that is eligible to transfer, each potential destination node j ($j = 1, \dots, n; j \neq i$) is compared with node i .

Definition 6.4.1. If $f_i^u > f_j^u + g^u$, then node i is said to be more heavily loaded than node j for a user u job.

We use the following proposition to determine a light node j relative to node i for a user u job, in the DGOS algorithm discussed below.

Proposition 6.4.2. If node i is more heavily loaded than node j for a user u job, then, $n_i^u > n_j^u$,

where

$$n_{ij}^u = \left[\frac{r_j}{r_i} \left(1 + \sum_{k=1}^m n_j^k \right)^2 + \frac{g^u}{r_i} \right]^{1/2} - \sum_{k=1, k \neq u}^m n_i^k - 1 \quad (6.101)$$

Proof: In Appendix D. □

We next describe the components of dynamic GOS (information policy, transfer policy and location policy).

1) *Information policy:* Each node i ($i = 1, \dots, n$) broadcasts the number of jobs of user j ($j = 1, \dots, m$) in its queue (*i.e.*, n_i^j 's) to all the other nodes. This state information exchange is done every P time units.

2) *Transfer policy:* A *threshold policy* is used to determine whether an arriving job should be processed locally or should be transferred to another node. A user j job arriving at node i will be eligible to transfer when the number of jobs of user j at node i is greater than some threshold denoted by T_i^j . Otherwise the job will be processed locally.

The thresholds T_i^j at a node i ($i = 1, \dots, n$) for each user j ($j = 1, \dots, m$) are calculated as follows:

Each node i broadcasts its arrival rates (ϕ_i^j) to all the other nodes every $P1$ time units where $P1 \gg P$. Using this information all the nodes execute GOS to determine the optimal loads (β_i^j). These optimal loads are then converted (using Little's law (Jain 1991), see proof of Proposition 6.4.1) into optimal number of jobs of user j that can be present at node i (thresholds T_i^j). The above calculated thresholds are fixed for an interval of $P1$ time units. This time interval can be adjusted based on the frequency of variation of the arrival rates at each node.

3) *Location policy:* The destination node for a user u job ($u = 1, \dots, m$) at node i that is eligible to transfer is determined based on the state information that is exchanged from the information policy. First, a node with the shortest marginal virtual delay for a user u job (lightest node) is determined. Next, it is determined whether the arriving job should be transferred based on the transfers the user u job already had.

(i) A node with the lightest load for a user u job is determined as follows: From Proposition 6.4.2, we have that if $n_i^u > n_{ij}^u$, then node i is said to be more heavily loaded than node j for a user u job. Else, if $n_i^u \leq n_{ij}^u$ then node i is said not to be heavily loaded than node j . Let $\delta_{ij}^u = n_i^u - n_{ij}^u$

and $\delta_i^u = \max_j \delta_{ij}^u$. If $\delta_i^u > 0$, then node j is the lightest loaded node for a user u job. Else, no light node is found and user u job will be processed locally.

(ii) Let c denote the number of times that a user u job has been transferred. Let ω ($0 < \omega \leq 1$) be a *weighting factor* used to prevent a job from being transferred continuously and let Δ ($\Delta > 0$) be a *bias* used to protect the system from instability by forbidding the load balancing policy to react to small load distinctions between the nodes. If $\omega^c \delta_i^u > \Delta$, then the job of user u will be transferred to node j . Otherwise, it will be processed locally.

We next describe the dynamic GOS (DGOS) algorithm.

DGOS algorithm:

For each node i ($i = 1, \dots, n$):

1. Jobs in the local queue will be processed with a mean service time of r_i .
2. If the time interval since the last state information exchange is P units, broadcast the n_i^u 's ($u = 1, \dots, m$) to all the other nodes.
3. If the time interval since the last threshold's calculation is $P1$ units,
 - i. Broadcast the ϕ_i^u 's ($u = 1, \dots, m$) to all the other nodes.
 - ii. Use the GOS algorithm to calculate β_i^u 's, $u = 1, \dots, m$.
 - iii. Use β_i^u 's, $u = 1, \dots, m$ to recalculate the thresholds (T_i^u , $u = 1, \dots, m$).

When a job of user u ($u = 1, \dots, m$) arrives with a mean inter-arrival time of a_i^u :

4. If $n_i^u \leq T_i^u$, then add the job to the local job queue. Go to Step 1.
5. Determine the lightest node j ($j = 1, \dots, n, j \neq i$) for the user u job as follows:
 - i. Calculate $\delta_{ij}^u = n_i^u - n_{ij}^u$ where n_{ij}^u is given by Proposition 6.4.2.
 - ii. Calculate $\delta_i^u = \max_j \delta_{ij}^u$.
 - iii. If $\delta_i^u > 0$, then node j is the lightest loaded node for the user u job.
 - iv. If $\omega^c \delta_i^u > \Delta$, where c is the number of transfers the job has already had, send the user u job to node j . Go to Step 1.
 - v. Add the user u job to the local queue. Go to Step 1.

6.4.2 Dynamic Non-cooperative Scheme (DNCOOPC)

The goal of DNCOOPC is to balance the workload among the nodes dynamically in order to obtain a user-optimal solution (*i.e.*, to minimize the expected response time of the individual users). We base the derivation of DNCOOPC on the static NCOOPC of Section 3.

We use the following proposition to express $\hat{f}_i^j(\beta_i)$ in eq. (6.92) in terms of the mean service time at node i (r_i) and the mean number of jobs of user j at node i (N_i^j , $j = 1, \dots, m$).

Proposition 6.4.3. Eq. (6.92) can be expressed in terms of r_i and N_i^j ($j = 1, \dots, m$) as

$$\hat{f}_i^j(\beta_i) = r_i \left(1 + \sum_{k=1}^m N_i^k\right) (1 + N_i^j) \quad (6.102)$$

Proof: Similar to Proposition 6.4.1. □

Rewriting eq. (6.93) in terms of ρ and ρ^{-j} , we have

$$\hat{g}^j(\lambda) = \frac{t(1 - \rho^{-j})}{(1 - \rho)^2}, \quad \rho < 1 \quad (6.103)$$

Let ρ' denote the utilization of the network at a given instant as in DGOS and $\rho^{-j'}$ denote the utilization of the network at a given instant excluding user j traffic. We next express $\hat{f}_i^j()$ of Proposition 6.4.3 and $\hat{g}^j()$ from eq. (6.103), which use the mean estimates of the system parameters, in terms of instantaneous variables.

$$\hat{f}_i^j = r_i \left(1 + \sum_{k=1}^m n_i^k\right) (1 + n_i^j) \quad (6.104)$$

$$\hat{g}^j = \frac{t(1 - \rho^{-j'})}{(1 - \rho')^2}, \quad \rho' < 1 \quad (6.105)$$

NCOOPC tries to balance the $\hat{f}_i^j(\beta_i)$ of each user j ($j = 1, \dots, m$) at all the nodes i ($i = 1, \dots, n$) statically whereas DNCOOPC tries to balance the \hat{f}_i^j of each user j at all the nodes i ($i = 1, \dots, n$) dynamically. For a user u job arriving at node i that is eligible to transfer, each potential destination node j ($j = 1, \dots, n; j \neq i$) is compared with node i .

Definition 6.4.2. If $\hat{f}_i^u > \hat{f}_j^u + \hat{g}^u$, then node i is said to be more heavily loaded than node j for a

user u job.

We use the following proposition to determine a light node j relative to node i for a user u job, in the DNCOOPC algorithm discussed below.

Proposition 6.4.4. If node i is more heavily loaded than node j for a user u job, then, $n_i^u > n_{ij}^u$, where

$$n_{ij}^u = \left[\frac{r_j}{r_i} \left(1 + \sum_{k=1}^m n_j^k \right) (1 + n_j^u) + \frac{g^u}{r_i} + \frac{\left(\sum_{k=1, k \neq u}^m n_i^k \right)^2}{4} \right]^{1/2} - \frac{\sum_{k=1, k \neq u}^m n_i^k}{2} - 1 \quad (6.106)$$

Proof: Similar to Proposition 6.4.2. □

We next describe the components of dynamic NCOOPC (information policy, transfer policy and location policy).

1) *Information policy:* Each node i ($i = 1, \dots, n$) broadcasts the number of jobs of user j ($j = 1, \dots, m$) in its queue (*i.e.*, n_i^j 's) to all the other nodes every P time units.

2) *Transfer policy:* A *threshold policy* is used to determine whether an arriving job should be processed locally or should be transferred to another node. A user j job arriving at node i will be eligible to transfer when the number of jobs of user j at node i is greater than some threshold T_i^j . Otherwise the job will be processed locally.

The thresholds T_i^j at a node i ($i = 1, \dots, n$) for each user j ($j = 1, \dots, m$) are calculated as follows:

Each node i broadcasts its arrival rates (ϕ_i^j) to all the other nodes every $P1$ time units where $P1 \gg P$. Using this information all the nodes execute NCOOPC to determine the optimal loads (β_i^j). These optimal loads are then converted (using Little's law (Jain 1991)) into optimal number of jobs of user j that can be present at node i (thresholds T_i^j). The above calculated thresholds are fixed for an interval of $P1$ time units. This time interval can be adjusted based on the frequency of variation of the arrival rates at each node.

3) *Location policy:* The destination node for a user u job ($u = 1, \dots, m$) at node i that is eligible to transfer is determined based on the state information that is exchanged from the information policy. First, a node with the lightest load for a user u job is determined. Next, whether the

arriving job should be transferred based on the transfers the user u job already had is determined.

(i) A node with the lightest load for a user u job is determined as follows: From Proposition 6.4.4, we have that if $n_i^u > n_{ij}^u$, then node i is said to be more heavily loaded relative to node j for a user u job. Else, if $n_i^u \leq n_{ij}^u$ then node i is said not to be heavily loaded relative to node j . Let $\delta_{ij}^u = n_i^u - n_{ij}^u$ and $\delta_i^u = \max_j \delta_{ij}^u$. If $\delta_i^u > 0$, then node j is the lightest loaded node for a user u job. Else, no light node is found and user u job will be processed locally.

(ii) Let c denote the number of times that a user u job has been transferred. Let ω ($0 < \omega \leq 1$) be a *weighting factor* used to prevent a job from being transferred continuously and let Δ ($\Delta > 0$) be a *bias* used to protect the system from instability by forbidding the load balancing policy to react to small load distinctions between the nodes. If $\omega^c \delta_i^u > \Delta$, then the job of user u will be transferred to node j . Otherwise, it will be processed locally.

We next describe the dynamic NCOOPC (DNCOOPC) algorithm.

DNCOOPC algorithm:

For each node i ($i = 1, \dots, n$):

1. Jobs in the local queue will be processed with a mean service time of r_i .
2. If the time interval since the last state information exchange is P units, broadcast the n_i^u 's ($u = 1, \dots, m$) to all the other nodes.
3. If the time interval since the last threshold's calculation is $P1$ units,
 - i. Broadcast the ϕ_i^u 's ($u = 1, \dots, m$) to all the other nodes.
 - ii. Use the NCOOPC algorithm to calculate β_i^u 's, $u = 1, \dots, m$.
 - iii. Use β_i^u 's, $u = 1, \dots, m$ to recalculate the thresholds (T_i^u , $u = 1, \dots, m$).

When a job of user u ($u = 1, \dots, m$) arrives with a mean inter-arrival time of a_i^u :

4. If $n_i^u \leq T_i^u$, then add the job to the local job queue. Go to Step 1.
5. Determine the lightest node j ($j = 1, \dots, n, j \neq i$) for the user u job as follows:
 - i. Calculate $\delta_{ij}^u = n_i^u - n_{ij}^u$ where n_{ij}^u is given by Proposition 6.4.4.

Table 6.1: System configuration

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

- ii. Calculate $\delta_i^u = \max_j \delta_{ij}^u$.
- iii. If $\delta_i^u > 0$, then node j is the lightest loaded node for the user u job.
- iv. If $\omega^c \delta_i^u > \Delta$, where c is the number of transfers the job has already had, send the user u job to node j . Go to Step 1.
- v. Add the user u job to the local queue. Go to Step 1.

6.5 Modeling Results

In this section, we evaluate the performance of the proposed dynamic schemes (DGOS and DNCOOPC) by comparing them with the static schemes (GOS and NCOOPC) using a computer model. The load balancing schemes are tested by varying some system parameters like the system utilization, overhead for job transfer, bias and the exchange period of state information.

We modeled a heterogeneous system consisting of 16 computers to study the effect of various parameters. The system has computers with four different service rates and is shared by 10 users. The system configuration is shown in Table 6.1. The first row contains the relative service rates of each of the four computer types. The relative service rate for computer C_i is defined as the ratio of the service rate of C_i to the service 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 third row shows the service rate of each computer type in the system. The performance metric used in our model is the *expected response time*.

System utilization (ψ) represents the amount of load on the system. It is defined as the ratio of the total arrival rate to the aggregate service rate of the system:

$$\psi = \frac{\Phi}{\sum_{i=1}^n \mu_i} \quad (6.107)$$

Table 6.2: Job arrival fractions q^j for each user

User	1	2	3-6	7	8-10
q^j	0.3	0.2	0.1	0.07	0.01

The total job arrival rate in the system Φ is determined by the system utilization ψ and the aggregate service rate of the system. We choose fixed values for system utilization and determined the total job arrival rate Φ . The job arrival rate for each user $\phi^j, j = 1, \dots, 10$ is determined from the total arrival rate as $\phi^j = q^j \Phi$, where the fractions q^j are given in Table 6.2. The job arrival rates of each user $j, j = 1, \dots, 10$ to each computer $i, i = 1, \dots, 16$, *i.e.*, the ϕ_i^j 's are obtained in a similar manner.

The mean service time of each computer $i, i = 1, \dots, 16$ and the mean inter-arrival time of a job of each user $j, j = 1, \dots, 10$ to each computer i that are used with the dynamic schemes are calculated from the mean service rate of each computer i and the mean arrival rate of a job of each user j to each computer i respectively. The utilization of the network at an instant (ρ') and the utilization of the network at an instant without user j traffic ($\rho^{-j'}$) are obtained by monitoring the network for the number of jobs of each user that are being transferred at that instant. The mean communication time for a job, t , is set to 0.01 sec. The overhead (OV) we use is defined as the percentage of service time that a computer has to spend to send or receive a job.

6.5.1 Effect of System Utilization

In Figure 6.2, we present the effect of system utilization (ranging from 10% to 90%) on the expected response time of the static and dynamic schemes studied when the overhead for job transfer is 0. The bias for job transfer (Δ) is set to 0.4 for both DGOS and DNCOOPC. The exchange period of state information (P) is 0.1 sec. The weighting factor for job transfer (ω) is set to 0.9 for both DGOS and DNCOOPC.

From Figure 6.2, it can be observed that at low to medium system utilization (load level), both the static and dynamic schemes show similar performance. But as the load level increases, the dynamic schemes, DGOS and DNCOOPC, provide substantial performance improvement over the static schemes. Also, the performance of DNCOOPC which minimizes the expected response time

of the individual users is very close to that of DGOS which minimizes the expected response time of the entire system. We note that these dynamic schemes are based on heuristics (by considering the solutions of the static schemes), and so the performance of DGOS may not be optimal (in terms of the overall expected response time) in all cases compared to DNCOOPC.

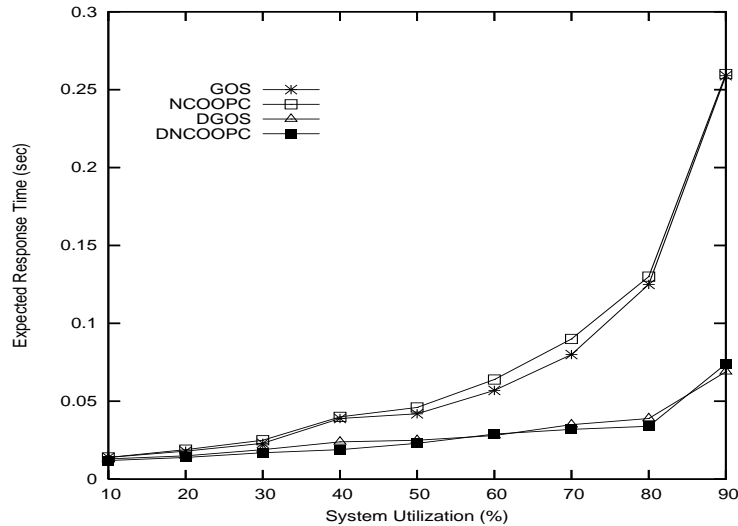


Figure 6.2: Variation of expected response time with system utilization ($OV = 0$)

In Figure 6.3, we present the expected response time for each user considering all the schemes at medium system utilization ($\psi = 60\%$). It can be observed that in the case of GOS and DGOS there are large differences in the users' expected response times. NCOOPC and DNCOOPC provides almost equal expected response times for all the users. Although, we plotted the users expected response times at load level $\psi = 60\%$, this kind of behavior of the load balancing schemes has been observed at all load levels. We say that NCOOPC and DNCOOPC are *fair* schemes (to all users), but GOS and DGOS are not *fair*.

From the above, we conclude that the DNCOOPC is a scheme which yields an allocation that makes almost as efficient use of the entire system resources as DGOS (as seen in Figure 6.2) and also tries to provide a user-optimal solution (as seen in Figure 6.3).

Figures 6.4 and 6.5 show the variation of expected response time with load level for the static and dynamic schemes taking the overhead for job transfer into consideration. In Figure 6.4, the overhead for sending and receiving a job is set to 5% of the mean job service time at a node and in Figure 6.5, the overhead is set to 10% of the mean job service time at a node. The other parameters (Δ, ω and P) are fixed as in Figure 6.2.

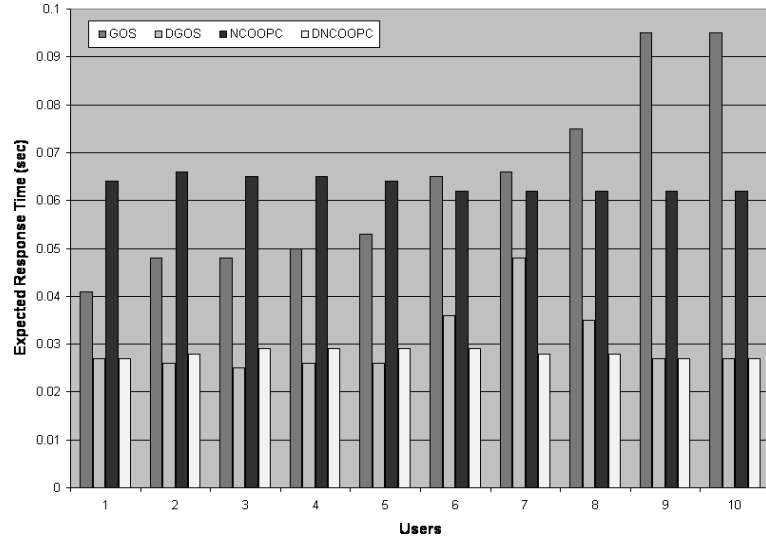


Figure 6.3: Expected response time for each user ($OV = 0$)

From Figure 6.4, it can be observed that at low and medium system loads the performance of static and dynamic schemes are similar. At high loads, although the expected response time of DGOS and DNCOOPC increases, the performance improvement is substantial compared to GOS and NCOOPC. As the overhead increases to 10% (Figure 6.5), at high system loads, the performance of dynamic schemes is similar to that of static schemes. This is because the dynamic schemes DGOS and DNCOOPC are more complex and the overhead costs caused by such complexity degrades their performance.

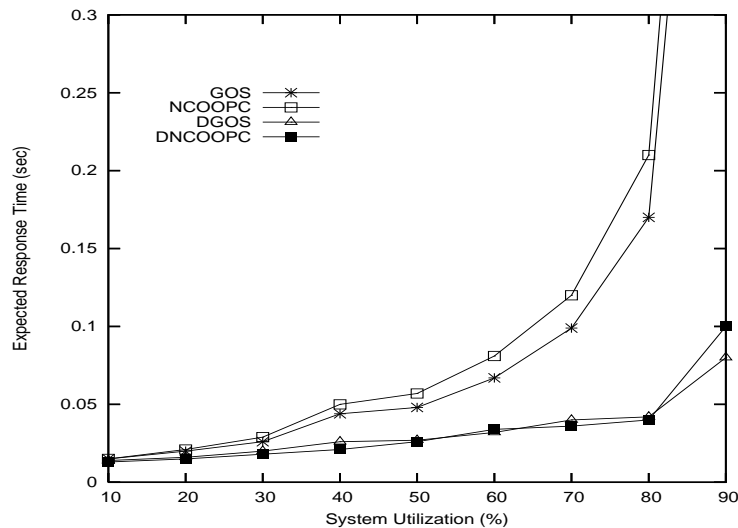


Figure 6.4: Variation of expected response time with system utilization ($OV = 5\%$)

From the above model results, it can be observed that at light to medium system loads (e.g. $\psi = 10\%$ to 50% in Figure 6.5), the performance of DGOS and DNCOOPC is insensitive to

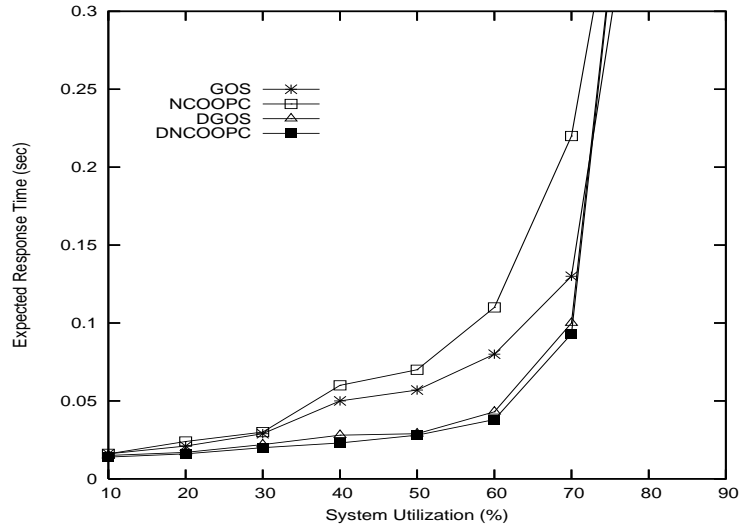


Figure 6.5: Variation of expected response time with system utilization ($OV = 10\%$)

overheads. But, at high system loads (e.g., $\psi = 80\%$ to 90% in Figure 6.5) the performance of DGOS and DNCCOOPC degrades with overhead and static schemes may be more efficient because of their less complexity.

6.5.2 Effect of Bias (Δ)

In Figure 6.6, we present the variation of expected response time with system utilization of DNCCOOPC for various biases. The overhead is assumed to be 5% . The other parameters are fixed as in Figure 6.2. It can be observed that as the bias increases, the expected response time of DNCCOOPC increases and for a high bias (e.g., $\Delta = 1$), the performance of DNCCOOPC is similar to that of GOS. The effect of bias on DGOS is similar.

6.5.3 Effect of Exchange Period (P)

In Figure 6.7, we present the variation of expected response time of DNCCOOPC with exchange period of system state information. The system utilization is fixed at 80% . The other parameters are fixed as in Figure 6.2. From Figure 6.7, it can be observed that the expected response time of DNCCOOPC increases with an increase in exchange period. This is because, for high values of P, outdated state information is exchanged between the nodes and optimal load balancing will not be done. The effect of exchange period on DGOS is similar.

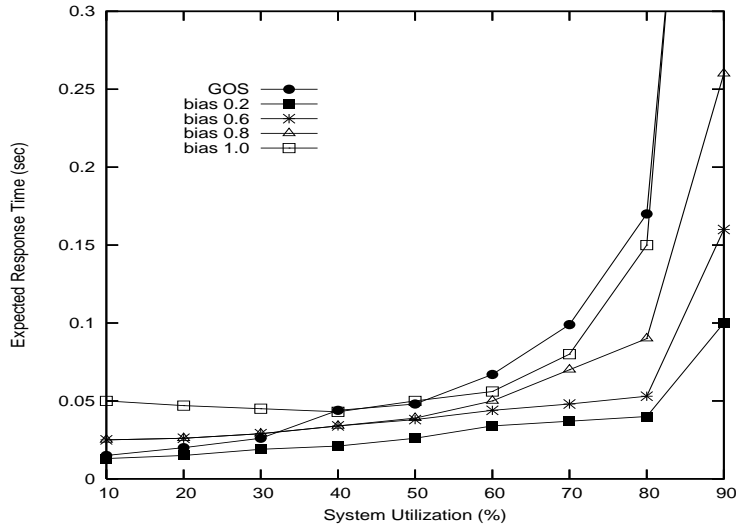


Figure 6.6: Variation of expected response time of DNCOOPC with system utilization for various biases ($OV = 5\%$)

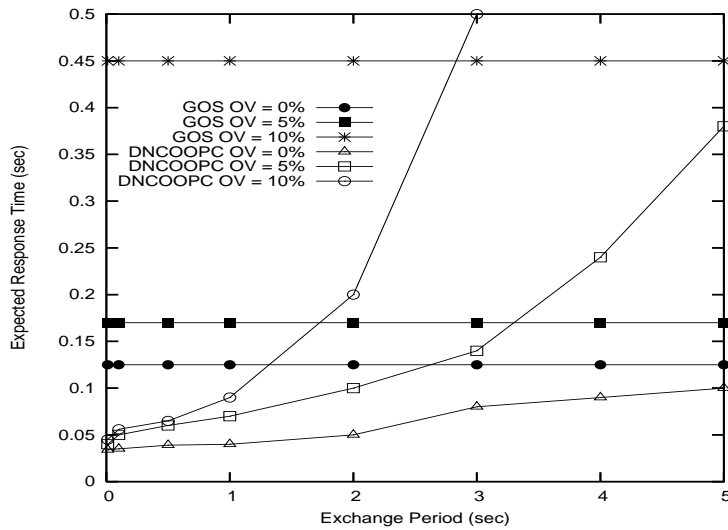


Figure 6.7: Variation of expected response time with exchange period ($\psi = 80\%$)

6.6 Summary

In this chapter, we proposed two dynamic load balancing schemes for multi-user jobs in heterogeneous distributed systems. DGOS tries to minimize the expected response time of the entire system, whereas DNCOOPC tries to minimize the expected response time of the individual users. These dynamic schemes use the number of jobs of the users in the queue at each node as state information. We run a computer model to compare the performance of the dynamic schemes with that of the static schemes (GOS and NCOOPC). It was observed that, at low communication overheads, both DGOS and DNCOOPC show superior performance over GOS and NCOOPC. Also, the performance

of DNCOOPC which tries to minimize the expected response time of the individual users is very close to that of DGOS which tries to minimize the expected response time of the entire system. Furthermore, DNCOOPC provides almost equal expected response times for all the users and so is a fair load balancing scheme. It was also observed that, as the bias, exchange period and the overheads for communication increases, both DGOS and DNCOOPC yield similar performance to that of the static schemes.

CHAPTER 7: Conclusions

In this chapter, we summarize our work, describe the contributions of this dissertation, and present future research directions.

7.1 Summary and Contributions of this Dissertation

We proposed job allocation or load balancing schemes for distributed systems and grid systems that provide fairness to the users and the users jobs. We used economic game theory that provides a suitable framework for characterizing such schemes. The fairness of allocation is an important factor in modern distributed systems and our schemes will be suitable for systems in which the fair treatment of the users' jobs is as important as other performance characteristics.

We considered a distributed computer system that consists of heterogeneous host computers (nodes) interconnected by a communication channel. Using cooperative game theory we proposed CCOOP static load balancing scheme for single-class job distributed systems. CCOOP is based on the Nash Bargaining Solution (NBS) which provides a Pareto optimal solution and also is a fair solution. Using a computer model, it was shown that CCOOP not only provides fairness to the jobs but also performs near the overall optimal scheme.

The single-class job distributed system model is then extended to a multi-class job distributed system model and pricing is included to model a grid system. Without taking the communication costs into account, we proposed two static price-based job allocation schemes (GOSP and NASHP) whose objective is to minimize the expected price of the grid users. The prices charged by the resource owners are obtained based on a pricing model using a bargaining game theory framework. GOSP tries to minimize the expected price of the entire grid community to provide a system-optimal solution whereas NASHP tries to minimize the expected price of the individual grid users to provide a fair solution. The performance of GOSP and NASHP is evaluated by running a grid model. GOSP is advantageous when the system optimum is required. But it is not fair to the grid users. NASHP not only provides fairness to the grid users but also performs near GOSP.

Considering the communication costs in the above grid system model, we proposed a static price-based job allocation scheme (NASHPC) whose objective is to provide fairness to the grid users.

This scheme is formulated as a non-cooperative game among the grid users who try to minimize the expected price of their own jobs. Using a grid model, the performance of NASHPC is compared with that of GOSPC (which tries to provide a system-optimal solution). It was observed that the performance of NASHPC is not only comparable with that of GOSPC in terms of the expected response time, but also provides an allocation which is fair (in terms of cost) to all the grid users.

NASHPC and GOSPC are then extended to dynamic load balancing schemes (DNCOOPC and DGOS) for multi-class job distributed systems. DGOS tries to minimize the expected response time of the entire system and DNCOOPC tries to minimize the expected response time of the individual users. Based on modeling results, it was observed that, at low communication overheads, both DGOS and DNCOOPC show superior performance over the static schemes. Also, DNCOOPC provides an allocation which is not only fair to the users but is also comparable to DGOS in terms of the overall expected response time.

7.2 Future Research Directions

We propose to study game theoretic models and algorithms for dynamic load balancing. Specifically our goals are:

(1) To propose dynamic load balancing schemes where the jobs arriving from various users to a node not only differ in their arrival rates but also differ in their processing times (*i.e.*, the jobs can be of different sizes).

(2) To propose new dynamic load balancing schemes. We intend to derive models and algorithms for dynamic load balancing based on dynamic games. The performance and fairness of these schemes will be analyzed.

(3) To implement the proposed games in a real distributed system.

7.2.1 Load balancing in Distributed Systems and Grids with Mechanism Design

Scheduling jobs for load balanced execution in heterogeneous distributed systems present new challenges. Difficulties arise due to geographic distribution and the heterogeneity of resources. The difficulty increases in distributed systems where resources belong to different self interested agents

or organizations. These agents may manipulate the load allocation algorithm to 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*. 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*.

Related work on load balancing based on mechanism design theory exist (Grosu and Chronopoulos 2004; Grosu and Chronopoulos 2003). In (Grosu and Chronopoulos 2004), a truthful mechanism that minimizes the overall expected response time of the system was proposed. In (Grosu and Chronopoulos 2003), a truthful mechanism based on cooperative game theory was proposed. We plan to develop truthful mechanisms for the non-cooperative algorithms and for other load balancing algorithms in multi-class job systems.

Since, self-interested agents are more common in grid systems, we also intend to implement the mechanism design protocols in conjunction with the job allocation schemes for grids proposed in the previous chapters.

APPENDICES

Appendix A

A.1 Proof of Theorem 3.4.1

The objective function for each player $f_i(X)$ (Definition 3.4.1) is convex and bounded below. The set of constraints is convex and compact. Thus, the conditions in Theorem 3.2.1 are satisfied and the result follows. \square

A.2 Proof of Theorem 3.4.2

The objective vector function $\{f_j\}$, $j \in 1, \dots, n+1$ (Definition 3.4.1) of the players is a one-to-one function of X . Thus, applying Theorem 3.2.1 the result follows. \square

A.3 Proof of Theorem 3.4.3

Let u_i and v_i denote the network traffic into node i and the network traffic out of node i respectively. From the balance of the total traffic in the network, we have

$$\sum_{i=1}^n u_i = \sum_{i=1}^n v_i \quad (7.108)$$

The load β_i on node i can then be written as

$$\beta_i = \phi_i + u_i - v_i \quad (7.109)$$

and the network traffic λ can be written as

$$\lambda = \sum_{i=1}^n v_i \quad (= \sum_{i=1}^n u_i) \quad (7.110)$$

Hence, the problem becomes:

$$\min E(u, v) = \left[\sum_{i=1}^n \ln D_i(\phi_i + u_i - v_i) + \ln G\left(\sum_{i=1}^n v_i\right) \right] \quad (7.111)$$

subject to

$$\beta_i = \phi_i + u_i - v_i \geq 0, \quad i = 1, \dots, n \quad (7.112)$$

$$-\sum_{i=1}^n u_i + \sum_{i=1}^n v_i = 0 \quad (7.113)$$

$$\beta_i = \phi_i + u_i - v_i < \mu_i, \quad i = 1, \dots, n \quad (7.114)$$

$$u_i \geq 0, \quad i = 1, \dots, n \quad (7.115)$$

$$v_i \geq 0, \quad i = 1, \dots, n \quad (7.116)$$

The objective function in eq. (7.111) is convex and the constraints are all linear and they define a convex polyhedron. This implies that the first-order Kuhn-Tucker conditions are necessary and sufficient for optimality (Luenberger 1984).

Let $\alpha, \delta_i \leq 0, \eta_i \leq 0, \psi_i \leq 0$ denote the Lagrange multipliers (Luenberger 1984) The Lagrangian is

$$L(u, v, \alpha, \delta, \eta, \psi) = E(u, v) + \alpha \left(-\sum_{i=1}^n u_i + \sum_{i=1}^n v_i \right) + \sum_{i=1}^n \delta_i (\phi_i + u_i - v_i) + \sum_{i=1}^n \eta_i u_i + \sum_{i=1}^n \psi_i v_i \quad (7.117)$$

Remark: We ignore the constraint in eq. (7.114) since all the associated multipliers will be zero if we introduce it in the Lagrangian.

The optimal solution satisfies the following Kuhn-Tucker conditions:

$$\frac{\partial L}{\partial u_i} = d_i(\phi_i + u_i - v_i) - \alpha + \delta_i + \eta_i = 0, \quad i = 1, \dots, n \quad (7.118)$$

$$\frac{\partial L}{\partial v_i} = -d_i(\phi_i + u_i - v_i) + g\left(\sum_{i=1}^n v_i\right) + \alpha - \delta_i + \psi_i = 0, \quad i = 1, \dots, n \quad (7.119)$$

$$\frac{\partial L}{\partial \alpha} = -\sum_{i=1}^n u_i + \sum_{i=1}^n v_i = 0 \quad (7.120)$$

$$\phi_i + u_i - v_i \geq 0, \quad \delta_i(\phi_i + u_i - v_i) = 0, \quad \delta_i \leq 0, \quad i = 1, \dots, n \quad (7.121)$$

$$u_i \geq 0, \quad \eta_i u_i = 0, \quad \eta_i \leq 0, \quad i = 1, \dots, n \quad (7.122)$$

$$v_i \geq 0, \quad \psi_i v_i = 0, \quad \psi_i \leq 0, \quad i = 1, \dots, n \quad (7.123)$$

In the following, we find an equivalent form of eqs. (7.118) - (7.123) in terms of β_i . By adding eqs. (7.118) and (7.119) we have, $-g(\sum v_i) = \eta_i + \psi_i$, $i = 1, \dots, n$. Since $g > 0$, either $\eta_i < 0$ or $\psi_i < 0$ (or both). Hence, from eqs. (7.122) and (7.123), for each i , either $u_i = 0$ or $v_i = 0$ (or both). We consider each case separately.

- *Case I: $u_i = 0, v_i = 0$:* Then, we have $\beta_i = \phi_i$. It follows from eq. (7.121) that $\delta_i = 0$. Substituting this into eqs. (7.118) and (7.119) gives

$$d_i(\beta_i) = \alpha - \eta_i \geq \alpha \quad (7.124)$$

$$d_i(\beta_i) = \alpha + g(\lambda) + \psi_i \leq \alpha + g(\lambda) \quad (7.125)$$

From the above, we have

$$\alpha \leq d_i(\beta_i) \leq \alpha + g(\lambda) \quad (7.126)$$

This case corresponds to neutral nodes.

- *Case II: $u_i = 0, v_i > 0$:* Then, from eq. (7.123), we have $\psi_i = 0$. We consider the following subcases.

- *Case II.1 $v_i < \phi_i$:* Then, we have $0 < \beta_i < \phi_i$. It follows from eq. (7.121) that $\delta_i = 0$. Substituting this into eqs. (7.118) and (7.119) gives

$$d_i(\beta_i) = \alpha - \eta_i \geq \alpha \quad (7.127)$$

$$d_i(\beta_i) = g(\lambda) + \alpha \quad (7.128)$$

This case corresponds to active sources.

- *Case II.2 $v_i = \phi_i$:* Then, we have $\beta_i = 0$ and eqs. (7.118) and (7.119) gives

$$d_i(\beta_i) = \alpha - \delta_i - \eta_i \geq \alpha \quad (7.129)$$

$$d_i(\beta_i) = \alpha + g(\lambda) - \delta_i \geq \alpha + g(\lambda) \quad (7.130)$$

Thus, we have

$$d_i(\beta_i) \geq \alpha + g(\lambda) \quad (7.131)$$

This case corresponds to idle sources.

- *Case III: $u_i > 0, v_i = 0$:* Then, we have $\beta_i > \phi_i$. It follows from eqs. (7.121) and (7.122) that $\delta_i = 0$ and $\eta_i = 0$. Substituting this into eq. (7.118), we have

$$d_i(\beta_i) = \alpha \quad (7.132)$$

This case corresponds to sinks.

Eq. (7.120) may be written in terms of β_i as

$$\sum_{i=1}^n \beta_i = \Phi \quad (7.133)$$

Using eqs. (7.128) and (7.132), the above equation becomes

$$\sum_{i \in S} d_i^{-1}(\alpha) + \sum_{i \in N} \phi_i + \sum_{i \in R_a} d_i^{-1}(\alpha + g(\lambda)) = \Phi \quad (7.134)$$

which is the total flow constraint. □

Appendix B

B.1 Proof of Theorem 4.4.1

We begin with the observation that the stability condition (eq. (4.45)) is always satisfied because of the fact that the total arrival rate (Φ) does not exceed the total processing rate of the distributed system. Thus we consider $D(\mathbf{s})$ problem with only two restrictions, eq. (4.43) and eq. (4.44).

We first show that $D(\mathbf{s})$ is a convex function in \mathbf{s} and that the set of feasible solutions defined by the constraints in eqs. (4.43) and (4.44) is convex.

From eq. (4.42) it can be easily shown that $\frac{\partial D(\mathbf{s})}{\partial s_{ji}} \geq 0$ and $\frac{\partial^2 D(\mathbf{s})}{\partial (s_{ji})^2} \geq 0$ for $i = 1, \dots, n$. This means that the Hessian of $D(\mathbf{s})$ is positive which implies that $D(\mathbf{s})$ is a convex function of the load fractions \mathbf{s} . The constraints are all linear and they define a convex polyhedron.

Thus, $D(\mathbf{s})$ involves minimizing a convex function over a convex feasible region and the first order Kuhn-Tucker conditions are necessary and sufficient for optimality.

Let $\alpha \geq 0$, $\eta_{ji} \geq 0$, $i = 1, \dots, n$, $j = 1, \dots, m$ denote the Lagrange multipliers. The Lagrangian is:

$$L(s_{11}, \dots, s_{mn}, \alpha, \eta_{11}, \dots, \eta_{mn}) = \sum_{j=1}^m \sum_{i=1}^n \frac{k_i p_{ji} \phi_j s_{ji}}{\Phi(\mu_i - \sum_{k=1}^m s_{ki} \phi_k)} - \alpha \left(\sum_{j=1}^m \sum_{i=1}^n s_{ji} - m \right) - \sum_{j=1}^m \sum_{i=1}^n \eta_{ji} s_{ji} \quad (7.135)$$

The Kuhn-Tucker conditions imply that s_{ji} , $j = 1, \dots, m$, $i = 1, \dots, n$ is the optimal solution to $D(\mathbf{s})$ if and only if there exists $\alpha \geq 0$, $\eta_{ji} \geq 0$, $j = 1, \dots, m$, $i = 1, \dots, n$ such that:

$$\frac{\partial L}{\partial s_{ji}} = 0 \quad (7.136)$$

$$\frac{\partial L}{\partial \alpha} = 0 \quad (7.137)$$

$$\eta_{ji} s_{ji} = 0, \eta_{ji} \geq 0, s_{ji} \geq 0, j = 1, \dots, m; i = 1, \dots, n \quad (7.138)$$

These conditions become:

$$\frac{k_i p_{ji} \phi_j \mu_i}{\Phi(\mu_i^j - s_{ji} \phi_j)^2} - \alpha - \eta_{ji} = 0, \quad j = 1, \dots, m; i = 1, \dots, n \quad (7.139)$$

$$\sum_{i=1}^n s_{ji} = 1, \quad j = 1, \dots, m \quad (7.140)$$

$$\eta_{ji} s_{ji} = 0, \eta_{ji} \geq 0, s_{ji} \geq 0, j = 1, \dots, m; i = 1, \dots, n \quad (7.141)$$

These are equivalent to:

$$\alpha = \frac{k_i p_{ji} \phi_j \mu_i}{\Phi(\mu_i^j - s_{ji} \phi_j)^2}, \text{ if } s_{ji} > 0; 1 \leq j \leq m; 1 \leq i \leq n \quad (7.142)$$

$$\alpha \leq \frac{k_i p_{ji} \phi_j \mu_i}{\Phi(\mu_i^j - s_{ji} \phi_j)^2}, \text{ if } s_{ji} = 0; 1 \leq j \leq m; 1 \leq i \leq n \quad (7.143)$$

$$\sum_{i=1}^n s_{ji} = 1, \quad s_{ji} \geq 0; \quad j = 1, \dots, m, \quad i = 1, \dots, n \quad (7.144)$$

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 ($\mu_1^j \geq \mu_2^j \geq \dots \geq \mu_n^j$), we have the following order on load fractions for each server: $s_{j1} \geq s_{j2} \geq \dots \geq s_{jn}$. This implies that may exist situations in which the slow computers have no jobs assigned to them by the servers. This means that there exist an index c_j ($1 \leq c_j \leq n$) so that $s_{ji} = 0$ for $i = c_j, \dots, n$ for each server.

From eq. (7.142) and based on the above claims we can obtain by summation the following equation for each server:

$$\sum_{i=1}^{c_j-1} \sqrt{k_i p_{ji} \phi_j \mu_i} = \sqrt{\alpha \Phi} \left(\sum_{i=1}^{c_j-1} \mu_i^j - \sum_{i=1}^{c_j-1} s_{ji} \phi_j \right) \quad (7.145)$$

Using eq. (7.143) the above equation becomes:

$$\sqrt{\alpha \Phi} = \frac{\sum_{i=1}^{c_j-1} \sqrt{k_i p_{ji} \phi_j \mu_i}}{\sum_{i=1}^{c_j-1} \mu_i^j - \sum_{i=1}^{c_j-1} s_{ji} \phi_j} \leq \frac{\sqrt{k_i p_{ji} \phi_j \mu_i}}{\mu_{c_j}^j} \quad (7.146)$$

This is equivalent to:

$$\mu_{c_j}^j \sum_{i=1}^{c_j} \sqrt{k_i p_{ji} \phi_j \mu_i} \leq \sqrt{k_i p_{ji} \phi_j \mu_i} \left(\sum_{i=1}^{c_j} \mu_i^j - \phi_j \right) \quad (7.147)$$

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

B.2 Proof of Theorem 4.4.2

The while loop in step 3 finds the minimum index c_j for which $\mu_{c_j}^j \leq \frac{\sqrt{k_i p_{ji} \mu_i} (\sum_{k=1}^{c_j} \mu_k^j - \phi_j)}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k}}$. In the same

loop, s_{ji} are set to zero for $i = c_j, \dots, n$. In step 4, s_{ji} is set equal to

$\frac{1}{\phi_j} \left(\mu_i^j - \sqrt{k_i p_{ji} \mu_i} \frac{\sum_{k=1}^{c_j} \mu_k^j - \phi_j}{\sum_{k=1}^{c_j} \sqrt{k_k p_{jk} \mu_k}} \right)$ for $i = 1, \dots, c_j - 1$. These are in accordance with Theorem 4.4.1.

Thus, the allocation $\{s_{j1}, \dots, s_{jn}\}$ computed by the BEST-FRACTIONS algorithm is the optimal solution for each server. \square

Appendix C

C.1 Proof of Theorem 5.3.1

We restate the problem introducing the variables u_i^j and v_i^j which denote the user j network traffic into node i and network traffic out of node i respectively. From the balance of the total traffic of user j in the network, we have

$$\lambda^j = \sum_{i=1}^n u_i^j = \sum_{i=1}^n v_i^j \quad (7.148)$$

The load β_i^j of user j on node i can then be written as

$$\beta_i^j = \phi_i^j + u_i^j - v_i^j, \quad i = 1, \dots, n \quad (7.149)$$

Using the above equations, the problem in eq. (5.59) becomes

$$\min_{u_i^j, v_i^j} D^j(u, v) = \left[\frac{1}{\phi^j} \sum_{i=1}^n \frac{k_i p_i^j (\phi_i^j + u_i^j - v_i^j)}{(\mu_i^j - (\phi_i^j + u_i^j - v_i^j))} + \frac{k_c p_c t \sum_{i=1}^n v_i^j}{\phi^j (g_{-j} - t \sum_{i=1}^n v_i^j)} \right] \quad (7.150)$$

subject to the following constraints:

$$-\sum_{i=1}^n u_i^j + \sum_{i=1}^n v_i^j = 0 \quad (7.151)$$

$$u_i^j - v_i^j + \phi_i^j \geq 0, \quad i = 1, \dots, n \quad (7.152)$$

$$u_i^j \geq 0, \quad i = 1, \dots, n \quad (7.153)$$

$$v_i^j \geq 0, \quad i = 1, \dots, n \quad (7.154)$$

We ignore the stability constraint (eq. (5.62)) because at Nash equilibrium it is always satisfied and the total arrival rate (Φ) does not exceed the total service rate of the system. The objective function in eq. (7.150) is convex and the constraints are all linear. This implies that the first-order Kuhn-Tucker conditions are necessary and sufficient for optimality (Reklaitis, Ravindran, and Ragsdell 1983). The total arrival rate of user j (ϕ^j) is constant.

Let $\alpha^j, \delta_i \leq 0, \psi_i \leq 0, \eta_i \leq 0$, denote the Lagrange multipliers (Reklaitis, Ravindran, and

Ragsdell 1983). The Lagrangian is

$$L(u^j, v^j, \alpha^j, \delta, \psi, \eta) = \phi^j D^j(u_i^j, v_i^j) + \alpha^j \left(\sum_{i=1}^n v_i^j - \sum_{i=1}^n u_i^j \right) + \sum_{i=1}^n \delta_i (u_i^j - v_i^j + \phi_i^j) + \sum_{i=1}^n \psi_i u_i^j + \sum_{i=1}^n \eta_i v_i^j \quad (7.155)$$

The optimal solution satisfies the following Kuhn-Tucker conditions:

$$\frac{\partial L}{\partial u_i^j} = f_i^j(\phi_i + u_i - v_i) - \alpha^j + \delta_i + \psi_i = 0, \quad i = 1, \dots, n. \quad (7.156)$$

$$\frac{\partial L}{\partial v_i^j} = -f_i^j(\phi_i + u_i - v_i) + g^j \left(\sum_{l=1}^n v_l \right) + \alpha^j - \delta_i + \eta_i = 0, \quad i = 1, \dots, n. \quad (7.157)$$

$$-\sum_{i=1}^n u_i^j + \sum_{i=1}^n v_i^j = 0 \quad (7.158)$$

$$\phi_i^j + u_i^j - v_i^j \geq 0, \quad \delta_i(\phi_i^j + u_i^j - v_i^j) = 0, \quad \delta_i \leq 0, \quad i = 1, \dots, n. \quad (7.159)$$

$$u_i^j \geq 0, \quad \psi_i u_i^j = 0, \quad \psi_i \leq 0, \quad i = 1, \dots, n. \quad (7.160)$$

$$v_i^j \geq 0, \quad \eta_i v_i^j = 0, \quad \eta_i \leq 0 \quad i = 1, \dots, n. \quad (7.161)$$

In the following, we find an equivalent form of eqs. (7.156) - (7.161) in terms of β_i . We consider two cases:

- *Case I:* $u_i^j - v_i^j + \phi_i^j = 0$: From eq. (7.149), we have $\beta_i^j = 0$ and since $\phi_i^j > 0$, it follows that $v_i^j > 0$. Eq. (7.161) implies $\eta_i = 0$. Then from eqs. (7.157) and (7.159), we get $f_i^j(\beta_i) = \alpha^j + g^j(\lambda) - \delta_i \geq \alpha^j + g^j(\lambda)$. This case corresponds to idle sources.

- *Case II:* $u_i^j - v_i^j + \phi_i^j > 0$: From eq. (7.159), we have $\delta_i = 0$.

- *Case II.1:* $v_i^j > 0$: Then, $0 < \beta_i^j < \phi_i^j$ and from eq. (7.161) we have $\eta_i = 0$. Eq. (7.157) implies,

$$f_i^j(\beta_i) = \alpha^j + g^j(\lambda) \quad (7.162)$$

This case corresponds to active sources.

- *Case II.2:* $v_i^j = 0$:

* *Case II.2.1: $w_i^j = 0$:* Then, $\beta_i^j = \phi_i^j$.

From eqs. (7.156) and (7.160), we have $f_i^j(\beta_i) = \alpha^j - \psi_i \geq \alpha_j$. From eqs. (7.157) and (7.161), we have $f_i^j(\beta_i) = \alpha^j + g^j(\lambda) + \eta_i \leq \alpha^j + g^j(\lambda)$. This case corresponds to neutral nodes.

* *Case II.2.2: $w_i^j > 0$:* Then, $\beta_i^j > \phi_i^j$.

From eq. (7.160), we have $\psi_i = 0$. Substituting this in eq. (7.156), we have

$$f_i^j(\beta_i) = \alpha^j \quad (7.163)$$

This case corresponds to sink nodes.

Eq. (7.158) may be written in terms of β_i^j as $\sum_{i=1}^n \beta_i^j = \phi_i^j$ and using the eqs. (7.162) and (7.163), this can be written as

$$\sum_{i \in S^j} (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j}) + \sum_{i \in N^j} \phi_i^j + \sum_{i \in R_a^j} (f_i^j)^{-1}(\beta_i |_{\beta_i^j = \alpha^j + g^j(\lambda)}) = \phi^j \quad (7.164)$$

which is the total flow constraint for user j . □

Appendix D

D.1 Proof of Proposition 6.4.1

The expected response time of a user j job processed at computer i in eq. (6.79) can be expressed in terms of r_i as

$$F_i^j(\beta_i) = \frac{r_i}{(1 - r_i \sum_{k=1}^m \beta_i^k)}$$

Using Little's law ($\sum_{k=1}^m N_i^k = \sum_{k=1}^m \beta_i^k F_i^j(\beta_i)$) (Jain 1991), the above equation can be written in terms of N_i^j , $j = 1, \dots, m$ as

$$F_i^j(\beta_i) = r_i \left(1 + \sum_{k=1}^m N_i^k\right) \quad (7.165)$$

Remark: Note that we assume the jobs from various users have the same processing time at a node.

The user j marginal node delay at node i , $f_i^j(\beta_i)$, is defined as

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} \sum_{k=1}^m \beta_i^k F_i^k(\beta_i)$$

which is equivalent to

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j} \sum_{k=1}^m \frac{\beta_i^k r_i}{(1 - r_i \sum_{l=1}^m \beta_i^l)}$$

Taking the partial derivative of the above equation with respect to β_i^j , we have

$$f_i^j(\beta_i) = \frac{r_i}{(1 - r_i \sum_{l=1}^m \beta_i^l)^2}$$

Using Little's law (Jain 1991), the above equation can be written in terms of N_i^j , $j = 1, \dots, m$ as

$$f_i^j(\beta_i) = \frac{r_i}{\left(1 - \frac{r_i \sum_{k=1}^m N_i^k}{F_i^j(\beta_i)}\right)^2}$$

which is equivalent to (using eq. (7.165))

$$f_i^j(\beta_i) = \frac{r_i}{\left(1 - \frac{r_i \sum_{k=1}^m N_i^k}{r_i(1 + \sum_{k=1}^m N_i^k)}\right)^2}$$

Thus, we have $f_i^j(\beta_i) = r_i(1 + \sum_{k=1}^m N_i^k)^2$ □

D.2 Proof of Proposition 6.4.2

For a user u job, node i is said to be more heavily loaded relative to node j if $f_i^u > f_j^u + g^u$.

Substituting eq. (6.99) for f_i^u and f_j^u , we have

$$r_i(1 + \sum_{k=1}^m n_i^k)^2 > r_j(1 + \sum_{k=1}^m n_j^k)^2 + g^u$$

which is equivalent to

$$(1 + n_i^u + \sum_{k=1, k \neq u}^m n_i^k)^2 > \frac{r_j}{r_i}(1 + \sum_{k=1}^m n_j^k)^2 + \frac{g^u}{r_i}$$

Thus, we have

$$n_i^u > \left[\frac{r_j}{r_i}(1 + \sum_{k=1}^m n_j^k)^2 + \frac{g^u}{r_i} \right]^{1/2} - \sum_{k=1, k \neq u}^m n_i^k - 1$$

Replacing the right-hand side of the above equation by n_{ij}^u , we have $n_i^u > n_{ij}^u$ where

$$n_{ij}^u = \left[\frac{r_j}{r_i}(1 + \sum_{k=1}^m n_j^k)^2 + \frac{g^u}{r_i} \right]^{1/2} - \sum_{k=1, k \neq u}^m n_i^k - 1$$

□

BIBLIOGRAPHY

- Akay, O. and K. Erciyes (2003). A dynamic load balancing model for a distributed system. *Math. and Computational Applications* 8(1-3), 353–360.
- Altman, E., H. Kameda, and Y. Hosokawa (2002). Nash equilibria in load balancing in distributed computer systems. *Intl. Game Theory Review* 4(2), 91–100.
- Anand, L., D. Ghose, and V. Mani (1999). Elisa: An estimated load information scheduling algorithm for distributed computing systems. *Computers and Mathematics with Applications* 37, 57–85.
- Anderson, T. E., D. E. Culler, D. A. Patterson, and the NOW team (1995, February). A case for now (networks of workstations). *IEEE Micro*. 15(1), 54–64.
- Basar, T. and G. J. Olsder (1998). *Dynamic Noncooperative Game Theory*. SIAM.
- Benmohammed-Mahieddine, K., P. M. Dew, and M. Kara (1994, June). A periodic symmetrically-initiated load balancing algorithm for distributed systems. In *Proc. of the 14th IEEE Intl. Conf. on Distributed Computing Systems*, pp. 616–623.
- Buyya, R., D. Abramson, J. Giddy, and H. Stockinger (2002, May). Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience (CCPE), Special Issue on Grid Computing Environments*, Wiley Press.
- Buyya, R., D. Abramson, and S. Venugopal (2005, March). The grid economy. *IEEE Special Issue on Grid Computing* 93(3), 698–714.
- Buyya, R., M. Murshed, D. Abramson, and S. Venugopal (2005, April). Scheduling parameter sweep applications on global grids: A deadline and budget constrained cost-time optimisation algorithm. *Intl. Jrnl of Software: Practice and Experience (SPE)* 35(5), 491–512.
- Cai, W., B. S. Lee, A. Heng, and L. Zhu (1997, December). 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*, pp. 383–389.
- Campos, L. M. and I. Scherson (2000, July). Rate of change load balancing in distributed and parallel systems. *Parallel Computing* 26(9), 1213–1230.

- Chao, K. M., R. Anane, J. H. Chen, and R. Gatward (2002). Negotiating agents in a market-oriented grid. In *Proc. of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany, pp. 436–437.
- Chavez, A., A. Moukas, and P. Maes (1997, February). *Challenger*: A multi-agent system for distributed resource allocation. In *Proc. of the Intl. Conf. on Autonomous Agents*, pp. 323–331.
- Choi, E. (2004). Performance test and analysis for an adaptive load balancing mechanism on distributed server cluster systems. *Future Generation Computer Systems* 20, 237–247.
- Chow, Y. C. and W. H. Kohler (1979, May). Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput. C-28*(5), 354–361.
- Corradi, A., L. Leonardi, and F. Zambonelli (1999, Jan.-March). Diffusive load-balancing policies for dynamic applications. *IEEE Concurrency* 7(1), 22–31.
- Cortes, A., A. Ripoll, M. A. Senar, and E. Luque (1999, February). Performance comparison of dynamic load balancing strategies for distributed computing. In *Proc. of the 32nd Hawaii Intl. Conf on System Sciences*, pp. 170–177.
- Cortes, A., A. Ripoll, M. A. Senar, P. Pons, and E. Luque (1999, January). On the performance of nearest-neighbors load balancing algorithms in parallel systems. In *Proc. of the 7th Euromicro Workshop on Parallel and Distributed Processing*, pp. 10.
- Dandamudi, S. P. (1998, July-Sept.). Sensitivity evaluation of dynamic load sharing in distributed systems. *IEEE Concurrency* 6(3), 62–72.
- Dixit, A. and S. Skeath (2004). *Games of Strategy*. New York: W. W. Norton & Company.
- El-Zoghdy, S. F., H. Kameda, and J. Li (Oct. 2002). A comparative study of static and dynamic individually optimal load balancing policies. In *Proc. of IASTED Intl. conf. on Networks, Parallel and Distributed Processing and Applications*, Tukuba, Japan.
- Elsasser, R., B. Monien, and R. Preis (2000, July). Diffusive load balancing schemes on heterogeneous networks. In *Proc. of the 12th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pp. 30–38.
- Esquivel, S. C., G. M. Leguizamon, and R. H. Gallard (1997, April). A hybrid strategy for load balancing in distributed systems environments. In *Proc. of the IEEE Intl. Conf. on Evolutionary*

Computation, pp. 127–132.

- Ferguson, D., C. Nikolaou, J. Sairamesh, and Y. Yemini (1996). Economic models for allocating resources in computer systems. In *Market based Control of Distributed Systems* (ed. S. Clearwater, World Scientific Press).
- Foster, I. and C. Kesselman (2004). *The Grid: Blueprint for a new Computing Infrastructure*. Morgan Kaufman.
- Fudenberg, D. and J. Tirole (1994). *Game Theory*. The MIT Press.
- Ghosh, B., S. Muthukrishnan, and M. H. Schultz (1996, June). 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*, pp. 72–81.
- Ghosh, P., K. Basu, and S. K. Das (2007, March). A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems. *IEEE Trans. on Parallel and Distributed Systems* 18(3), 289–306.
- Ghosh, P., K. Basu, and S. K. Das (Dec. 2005). Cost-optimal job allocation schemes for bandwidth-constrained distributed computing systems. In *Proceedings of 12th Annual IEEE International Conference on High Performance Computing (HiPC)*, Goa, India.
- Ghosh, P., N. Roy, K. Basu, and S. Das (2004). A game theory based pricing strategy for job allocation in mobile grids. In *Proc. of the 18th IEEE International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, USA, pp. 26–30.
- Ghosh, P., N. Roy, S. K. Das, and K. Basu (2005, Nov). A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework. *Journal of Parallel and Distributed Computing* 65(11), 1366–1383.
- Grosu, D. and A. T. Chronopoulos (2002). A game-theoretic model and algorithm for load balancing in distributed systems. In *Proc. of the 16th IEEE International Parallel and Distributed Processing Symposium*, Ft Lauderdale, Florida, USA, pp. 146–153.
- Grosu, D. and A. T. Chronopoulos (2003, April). A truthful mechanism for fair load balancing in distributed systems. In *Proc. of the 2nd IEEE Intl. Symp. on Network Computing and Applications (NCA'03)*, pp. 289–296.

- Grosu, D. and A. T. Chronopoulos (2004, Feb.). Algorithmic mechanism design for load balancing in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics - Part B* 34(1), 77–84.
- Grosu, D. and A. T. Chronopoulos (2005, Sep.). Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing* 65(9), 1022–1034.
- Grosu, D., A. T. Chronopoulos, and M. Y. Leung (April 2002). Load balancing in distributed systems: An approach using cooperative games. In *Proc. of the 16th IEEE Intl. Parallel and Distributed Processing Symp.*, Ft Lauderdale, Florida, USA, pp. 52–61.
- Grosu, D. and A. Das (Nov. 2004). Auction-based resource allocation protocols in grids. In *Proc. of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, Cambridge, Massachusetts, USA, pp. 20–27.
- Han, C. C., K. G. Shin, and S. K. Yun (2000, September). On load balancing in multicomputer/distributed systems equipped with circuit or cut-through switching capability. *IEEE Trans. Comput.* 49(9), 947–957.
- Harsanyi, J. (1967, Nov.). Games with incomplete information played by bayesian players, part i, the basic model. *Management Science* 14(3), 159–182.
- Harsanyi, J. (1968, Jan.). Games with incomplete information played by bayesian players, part ii, bayesian equilibrium points. *Management Science* 14(5), 320–334.
- Hui, C. C. and S. T. Chanson (1999, July-Sept.). Improved strategies for dynamic load balancing. *IEEE Concurrency* 7(3), 58–67.
- Inoie, A., H. Kameda, and C. Touati (Dec. 2004). Pareto set, fairness, and nash equilibrium: A case study on load balancing. In *Proc. of the 11th Intl. Symp. on Dynamic Games and Applications*, Tucson, Arizona, pp. 386–393.
- Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience.
- Kameda, H., J. Li, C. Kim, and Y. Zhang (1997). *Optimal Load Balancing in Distributed Computer Systems*. Springer Verlag, London.
- Kim, C. and H. Kameda (1990). Optimal static load balancing of multi-class jobs in a distributed computer system. In *Proc. of the 10th International Conference on Distributed Computing*

Systems, pp. 562–569.

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

Kleinrock, L. (1975). *Queueing Systems - Volume 1: Theory*. John Wiley and Sons.

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

Kwok, Y. K., K. Hwang, and S. Song (2007, May). Selfish grids: Game-theoretic modeling and nas/psa benchmark evaluation. *IEEE Trans. on Parallel and Distributed Systems* 18(5), 621–636.

Kwok, Y. K., S. Song, and K. Hwang (May 2005). Selfish grid computing: Game-theoretic modeling and nas performance results. In *Proc. of the CCGrid*, Cardiff, U.K.

Larson, K. and T. Sandholm (2002). An alternating offers bargaining model for computationally limited agents. *AAMAS'02, Bologna, Italy*.

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

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

Lee, S. H., T. W. Kang, M. S. Ko, G. S. Chung, J. M. Gil, and C. S. Hwang (1997, May). 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, pp. 302–307.

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

Luce, R. D. and H. Raiffa (1957). *Games and Decisions*. Wiley, New York.

Luenberger, D. G. (1984). *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Mass.

- Mas-Collel, A., M. D. Whinston, and J. R. Green (1995). *Microeconomic Theory*. Oxford Univ. Press, New York.
- Mitzenmacher, M. (1997, June). On the analysis of randomized load balancing schemes. In *Proc. of the 9th ACM Intl. Symp. on Parallel Algorithms and Architectures*, pp. 292–301.
- Muthoo, A. (1999). *Bargaining Theory with Applications*. Cambridge Univ. Press, Cambridge, U.K.
- Nash, J. (1950, April). The bargaining problem. *Econometrica* 18(2), 155–162.
- Nash, J. (1951, September). Non-cooperative games. *Ann. Math.* 54(2), 286–295.
- Osborne, M. J. (2004). *An Introduction to Game Theory*. Oxford University Press.
- Osborne, M. J. and A. Rubinstein (1990). *Bargaining and Markets*. Academic Press Inc.
- Owen, G. (1982). *Game Theory*. Academic Press.
- Penmatsa, S. and A. T. Chronopoulos (2005). Job allocation schemes in computational grids based on cost optimization. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium, Joint Workshop on HPGC and HIPS*, Denver, Colorado, USA.
- Penmatsa, S. and A. T. Chronopoulos (April 2006a). Cooperative load balancing for a network of heterogeneous computers. In *Proc. of the 20th IEEE Intl. Parallel and Distributed Processing Symposium, 15th Heterogeneous Computing Workshop*, Rhodes Island, Greece.
- Penmatsa, S. and A. T. Chronopoulos (April 25-29, 2006b). Price-based user-optimal job allocation scheme for grid systems. In *Proc. of the 20th IEEE Intl. Parallel and Distributed Proc. Symp., 3rd High Performance Grid Computing Workshop*, Rhodes Island, Greece.
- Penmatsa, S. and A. T. Chronopoulos (March 26-30, 2007). Dynamic multi-user load balancing in distributed systems. In *Proc. of the 21st IEEE Intl. Parallel and Distributed Proc. Symp.*, Long Beach, California.
- Rabani, Y., A. Sinclair, and R. Wanka (1998, November). 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*, pp. 694–703.
- Reklaitis, G. V., A. Ravindran, and K. M. Ragsdell (1983). *Engineering Optimization: Methods and Applications*. Wiley-Interscience.

- Ross, K. W. and D. D. Yao (1991, July). Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM* 38(3), 676–690.
- Roughgarden, T. (July 2001). Stackelberg scheduling strategies. In *Proc. of the 33rd Annual ACM Symp. on Theory of Computing*, pp. 104–113.
- Shivaratri, N. G., P. Krueger, and M. Singhal (1992, December). Load distribution for locally distributed systems. *IEEE Computer* 8(12), 33–44.
- Stefanescu, A. and M. V. Stefanescu (1984). The arbitrated solution for multi-objective convex programming. *Rev. Roum. Math. Pure Appl.* 29, 593–598.
- Tang, X. and S. T. Chanson (2000). Optimizing static job scheduling in a network of heterogeneous computers. In *Proc. of the International Conf. on Parallel Processing*, pp. 373–382.
- Tantawi, A. N. and D. Towsley (1985, April). Optimal static load balancing in distributed computer systems. *Journal of the ACM* 32(2), 445–465.
- Winoto, P., G. McCalla, and J. Vassileva (2002). An extended alternating-offers bargaining protocol for automated negotiation in multi-agent systems. In *Proc. of the 10th International Conference on Cooperative Information Systems*, Irvine, CA, USA, pp. 179–194.
- Yaiche, H., R. R. Mazumdar, and C. Rosenberg (2000, October). A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE / ACM Trans. Networking* 8(5), 667–678.
- Yu, D. and T. G. Robertazzi (Nov. 2003). Divisible load scheduling for grid computing. In *Proc. of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, USA.
- Zeng, Z. and V. Bharadwaj (2004). Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks. *Computer Comm.* 27, 679–694.
- Zeng, Z. and B. Veeravalli (2006, Nov). Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks. *IEEE Transactions on Computers* 55(11), 1410–1422.
- Zhang, Y., H. Kameda, and S. L. Hung (1997, March). Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems. *IEE Proc. Computers and Digital*

Techniques 144(2), 100–106.

VITA

Satish Penmatsa was born in Goraganamudi, India on February 20, 1978, the son of Sri Hari Raju Penmatsa and Vijaya Lakshmi Penmatsa. He received the Bachelor of Technology in Computer Science from Andhra University, India in 2000 and Master of Science in Computer Science from The University of Texas at San Antonio in 2003. From 2003 to 2007 he was a Doctoral student in the Department of Computer Science at The University of Texas at San Antonio. His research interests include Parallel and Distributed Systems, High Performance Computing, Grid Computing, Game Theory, and Science and Engineering Applications. He has published 2 journal and 8 conference and workshop papers on these topics. He also served as a reviewer for Parallel Computing Journal, IEEE Transactions on Parallel and Distributed Systems, IEEE International Symposium on Network Computing and Applications, and Workshop on Incentive Based Computing. He is a student member of the IEEE, IEEE Computer Society, and the ACM.