Available online at www.sciencedirect.com
SCIENCE DIRECT®
J. Parallel Distrib. Comput. 65 (2005) 1022–1034
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Noncooperative load balancing in distributed systems

Daniel Grosu[a], Anthony T. Chronopoulos[b,*]

[a] *Department of Computer Science, Wayne State University, 5143 Cass Avenue, Detroit, MI 48202, USA*
[b] *Department of Computer Science, University of Texas at San Antonio, 6900 N. Loop 1604 West, San Antonio, TX 78249, USA*

Received 12 March 2003; received in revised form 12 October 2004; accepted 9 May 2005
Available online 22 June 2005

Abstract

In this paper, we present a game theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. We formulate the static load balancing problem in heterogeneous distributed systems as a noncooperative game among users. For the proposed noncooperative load balancing game, we present the structure of the Nash equilibrium. Based on this structure we derive a new distributed load balancing algorithm. Finally, the performance of our noncooperative load balancing scheme is compared with that of other existing schemes. The main advantages of our load balancing scheme are the distributed structure, low complexity and optimality of allocation for each user.

Keywords: Distributed systems; Static load balancing; Game theory; Nash equilibrium; Noncooperative games; Performance evaluation

1. Introduction

In recent years, heterogeneous distributed computing systems have become the main platforms for the execution of distributed applications. In these systems applications are submitted by a large number of users that compete for the shared heterogeneous resources (computers, storage communication links, etc.) [10,12,38]. Thus, a distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases the load balancing problem becomes important. The purpose of load balancing is to improve the performance of a distributed system through an appropriate distribution of the application load. A general formulation of this problem is as follows: given a large number of jobs, find the allocation of jobs to computers optimizing a given objective function (e.g. total execution time).

There is a large body of literature on load balancing and all the proposed load balancing schemes can be broadly characterized as static and dynamic [5]. Static load balancing schemes use a priori knowledge of the applications and statistical information about the system whereas dynamic load balancing schemes base their decision making process on the current state of the system. A good load balancing scheme needs to be general, stable, scalable, and to add a small overhead to the system. These requirements are interdependent, for example a general load balancing scheme may add a large overhead to the system, while an application specific load balancing scheme may have a very small overhead. Since the focus of this paper is on the static load balancing problem, we will not discuss the details related to dynamic load balancing. For further details on dynamic load balancing, see [25,37].

The static load balancing problem can be solved by employing one of the following three approaches:

1. *Global approach*: In this case there is only one decision maker that optimizes the response time of the entire system over all jobs and the operating point is called *social* (*overall*) *optimum*. This is the classical approach and has been studied extensively using different techniques such as nonlinear optimization [16,17,21,22,29,35,36] and

* Corresponding author. Fax: +1 210 458 4437.
E-mail addresses: dgrosu@cs.wayne.edu (D. Grosu), atc@cs.utsa.edu (A.T. Chronopoulos).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.05.001

polymatroid optimization [32]. All existing schemes using the global approach are not based on game theoretic models.

2. *Cooperative approach*: In this case there are several decision makers (e.g. jobs, computers) that cooperate in making the decisions such that each of them will operate at its optimum. Decision makers have complete freedom of preplay communication to make joint agreements about their operating points. This situation can be modeled as a cooperative game and game theory offers a suitable modeling framework [11].
3. *Noncooperative approach*: In this case there are several decision makers (e.g users, jobs) that are not allowed to cooperate in making decisions. Each decision maker optimizes its own response time independently of the others and they all eventually reach an equilibrium. This situation can be viewed as a noncooperative game among decision makers. The equilibrium is called *Nash equilibrium* [11,28] and it can be obtained by a distributed noncooperative policy. At the Nash equilibrium a decision maker cannot receive any further benefit by changing its own decision. If the number of decision makers is not finite the Nash equilibrium reduces to the *Wardrop equilibrium* [15].

In this paper, we investigate the noncooperative approach and propose a new model and a load balancing algorithm based on a noncooperative game among users.

1.1. Related work

There exist only few studies on game theoretic models and algorithms for load balancing in distributed systems. Kameda et al. [15] studied noncooperative games and derived load balancing algorithms for computing the Wardrop equilibrium in single class and multi-class job distributed systems. Roughgarden [33] formulated the scheduling problem as a Stackelberg game. In this type of noncooperative game one player acts as a leader and the rest as followers. He showed that it is NP-hard to compute the optimal Stackelberg strategy and presents efficient algorithms to compute strategies inducing near-optimal solutions. Grosu and Chronopoulos [13] formulated the static load balancing problem in single class job distributed systems as a cooperative game among computers and derived a load balancing scheme based on the Nash Bargaining Solution [27].

Routing traffic in networks is a closely related problem that has received more attention. Economides and Silvester [9] considered a noncooperative routing problem for two classes of packets. The first class objective is to minimize its *average packet delay*, while the other class objective is to minimize its *blocking probability*. They derived a routing policy for a two server system and presented the strategy and the performance of each class. They also studied in [8] the Nash equilibrium of the routing problem for two classes of users and two servers. Orda et al. [30] studied a noncooperative game in a network of parallel links with convex cost functions. They studied the existence and uniqueness of the Nash equilibrium. Altman et al. [1] investigated the same problem in a network of parallel links with linear cost functions. Korilis et al. [19] considered the capacity allocation problem in a network shared by noncooperative users. They studied the structure and the properties of Nash equilibrium for a routing game with M/M/1-type cost functions (i.e. functions that characterize the expected delay in a queue with Poisson arrivals and exponentially distributed processing times [18]). An important line of research was initiated by Koutsoupias and Papadimitriou [20], who considered a noncooperative routing game and proposed the ratio between the worst possible Nash equilibrium and the overall optimum as a measure of effectiveness of the system. Mavronicolas and Spirakis [24] derived tight bounds on coordination ratio in the case of fully mixed strategies where each user assigns its traffic with nonzero probability to every link. Roughgarden and Tardos [34] showed that in a network in which the link cost functions are linear the flow at Nash equilibrium has total latency at most $\frac{4}{3}$ that of the overall optimal flow. They also showed that if the link cost functions are assumed to be only continuous and nondecreasing the total latency may be arbitrarily larger than the minimum possible total latency.

1.2. Motivation and contribution

Most of the previous studies on static load balancing considered as their main objective the minimization of overall expected response time. This is difficult to achieve in distributed systems where there is no central authority controlling the allocation and users are free to act in a selfish manner. Our goal is to find a formal framework for characterizing user-optimal allocation schemes in distributed systems. The framework was provided by noncooperative game theory which has been applied to routing and flow control problems in networks. Kameda et al. [15] used this framework and obtained the Wardrop equilibrium for a noncooperative load balancing game among jobs. In this paper, we formulate the load balancing problem in distributed systems as a *noncooperative game among users*. We adopt the Nash equilibrium as the solution of this game. Nash equilibrium provides a user-optimal operation point for the distributed system. We give a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compare the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user (not the overall system optimum) in the distributed system.

1.3. Organization

The paper is structured as follows. In Section 2, we present the system model and we introduce our load balancing non-

cooperative game. In Section 3, we derive a greedy distributed algorithm for computing the Nash equilibrium for our load balancing game. In Section 4, the performance of our load balancing scheme is compared with those of other existing schemes. In Section 5, we draw conclusions and present future directions.

2. Load balancing as a noncooperative game among users

We consider a distributed system that consists of n heterogeneous computers shared by m users. Each computer is modeled as an M/M/1 queueing system (i.e. Poisson arrivals and exponentially distributed processing times) [18] and is characterized by its average processing rate $\mu_i, i = 1, \ldots, n$. Jobs are generated by user j with an average rate ϕ_j, and $\Phi = \sum_{j=1}^{m} \phi_j$ is the total job arrival rate in the system. The total job arrival rate Φ must be less than the aggregate processing rate of the system (i.e. $\Phi < \sum_{i=1}^{n} \mu_i$). The system model is presented in Fig. 1. The problem faced by users is to decide on how to distribute their jobs to computers such that they will operate optimally. Thus, user j ($j = 1, \ldots, m$) must find the fraction s_{ji} of its workload that is assigned to computer i ($\sum_{i=1}^{n} s_{ji} = 1$ and $0 \leqslant s_{ji} \leqslant 1$, $i = 1, \ldots, n$) such that the expected execution time of its jobs is minimized. Once s_{ji} are determined by using the algorithm proposed in this paper, user j sends jobs to computer i at a rate given by $s_{ji}\phi_j$ (in jobs/s). Note that s_{ji} are nondimensional.

We formulate the above problem as a noncooperative game among users under the assumption that users are 'selfish'. This means that they minimize the expected response time of their own jobs. In the following, we first present the notations we use and then define the noncooperative load balancing game.

Let s_{ji} be the fraction of workload that user j sends to computer i. The vector $\mathbf{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$ is called the *load balancing strategy* of user j, $j = 1, \ldots, m$. The vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ is called the *strategy profile* of the load balancing game.

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

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

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

$$D_j(\mathbf{s}) = \sum_{i=1}^{n} s_{ji} F_i(\mathbf{s}) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i - \sum_{k=1}^{m} s_{ki}\phi_k}. \tag{2}$$

The goal of user j is to find a feasible load balancing strategy $\mathbf{s}_j$ such that $D_j(\mathbf{s})$ is minimized. The decision of user j depends on the load balancing decisions of the other users since $D_j(\mathbf{s})$ is a function of $\mathbf{s}$.

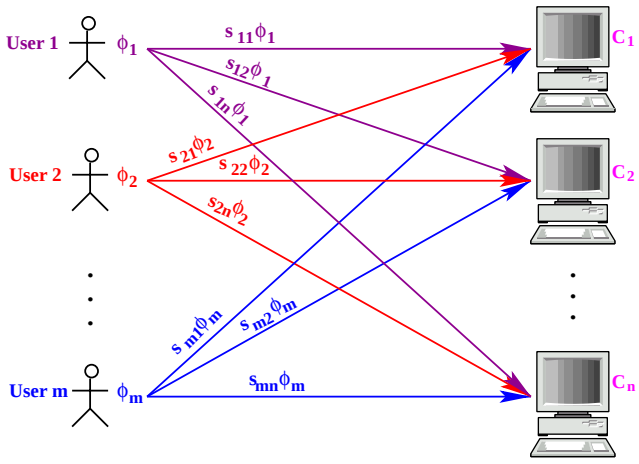

Fig. 1. The distributed system model.

Definition 2.1 (*Feasible strategy profile*). A *feasible load balancing strategy profile* is a strategy profile $\mathbf{s}$ that satisfies the following restrictions:

(i) *Positivity*: $s_{ji} \geqslant 0$, $i = 1, \ldots, n$, $j = 1, \ldots, m$;
(ii) *Conservation*: $\sum_{i=1}^{n} s_{ji} = 1$, $j = 1, \ldots, m$;
(iii) *Stability*: $\sum_{j=1}^{m} s_{ji}\phi_j < \mu_i$, $i = 1, \ldots, n$.

Now we define our noncooperative load balancing game.

Definition 2.2 (*Noncooperative load balancing game*). The *Noncooperative load balancing 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 load balancing strategies.
(iii) *Preferences*: Each user's preferences are represented by its expected response time (D_j). Each user j prefers the strategy profile $\mathbf{s}$ to the strategy profile $\mathbf{s}'$ if and only if $D_j(\mathbf{s}) < D_j(\mathbf{s}')$.

In order to obtain a load balancing scheme for the distributed system we need to solve the above game. The most commonly used solution concept for such games is that of Nash equilibrium [11], which we consider here.

Definition 2.3 (*Nash equilibrium*). A *Nash equilibrium* of the load balancing game defined above is a strategy profile $\mathbf{s}$ such that for every user j ($j = 1, \ldots, m$):

$$\mathbf{s}_j \in \arg\min_{\tilde{\mathbf{s}}_j} D_j(\mathbf{s}_1, \ldots, \tilde{\mathbf{s}}_j, \ldots, \mathbf{s}_m). \tag{3}$$

Nash equilibrium for our load balancing game is a strategy profile with the property that no user can decrease its expected job execution time by choosing a different load

balancing strategy given the other user's load balancing strategies. In other words a strategy profile $\mathbf{s}$ is a Nash equilibrium if no user can benefit by deviating unilaterally from its load balancing strategy to another feasible one. For our load balancing game there exists a unique Nash equilibrium because the expected response time functions are continuous, convex and increasing [30].

Remark. In general Nash equilibria are defined in terms of mixed strategies which are probability distributions over the set of pure strategies. In this paper, we are interested only in pure strategy equilibria. In pure strategy equilibria a user (player) chooses a unique strategy from the set of available strategies, whereas in mixed strategy equilibria he chooses a probability distribution over the set of strategies available to him [31].

Similar games were studied in the context of flow control and routing in networks. Orda et al. [30] proved that if the expected response time functions are continuous, convex and increasing there exists a unique Nash equilibrium for the game. The closest work to our study is that of Korilis et al. [19] in which a similar game is studied in the context of capacity allocation in networks. They studied the structure and properties of the Nash equilibrium for the game. Here, we are interested in finding a way to determine the Nash equilibrium for our load balancing noncooperative game.

In order to determine a solution for our load balancing game we consider an alternative definition of the Nash equilibrium. Nash equilibrium can be defined as a strategy profile for which every user's load balancing strategy is a *best reply* to the other users strategies. This best reply for a user will provide a minimum expected response time for that user's jobs given the other users' strategies. This definition gives us a method to determine the structure of the Nash equilibrium for our load balancing game. First, we determine the best reply strategies $\mathbf{s}_j$ for each user j, then we find a strategy profile $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ for which $\mathbf{s}_j$ is the best reply of user j, for $j = 1, 2, \ldots, m$.

First, we determine the best reply of user j, $j = 1, 2, \ldots, m$ which is the strategy profile that obtains the minimum expected response time for user j jobs with respect to the other users strategies. Let $\mu_i^j = \mu_i - \sum_{k=1,k\neq j}^{m} s_{ki}\phi_k$ be the *available processing* rate at processor i as seen by user j. The problem of computing the best reply strategy of user j ($j = 1, \ldots, m$) reduces to computing the optimal strategy for a system with one user and n computers having μ_i^j ($i = 1, \ldots, n$) as processing rates and ϕ_j as the user's job arrival rate in the system.

This can be translated into the following optimization problem (BEST-REPLY$_j$):

$$\min_{\mathbf{s}_j} D_j(\mathbf{s}) \tag{4}$$

subject to the constraints

$$s_{ji} \geqslant 0, \quad i = 1, \ldots, n, \tag{5}$$

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

$$\sum_{k=1}^{m} s_{ki}\phi_k < \mu_i, \quad i = 1, \ldots, n. \tag{7}$$

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

There exist some algorithms for finding the solution for similar optimization problems using different objective functions based on Lagrange multipliers. One was proposed by Tantawi and Towsley [36] but it is complex and involves a numerical method for solving a nonlinear equation. Ni and Hwang [29] studied a similar problem for a system with preassigned arrival rates at each computer, but they considered a different objective function from ours. Another one which inspired our approach was proposed by Tang and Chanson [35]. They considered a system were there is only one user that has to assign jobs to computers such that the job execution time for all jobs is minimized. We extend the work in [35] considering a different model in which the influence of the other users' decisions on the optimum is taken into account. Thus, we use the same objective function but we use different constraints. Here, we first characterize the best reply strategy of each user and then propose an algorithm for computing this strategy considering our model. The algorithm will be used in Section 3 to devise a distributed algorithm for computing the Nash equilibrium for our load balancing game.

The best reply strategy of user j which is the solution of BEST-REPLY$_j$ is given in the following theorem.

Theorem 2.1 (*BEST-REPLY$_j$ solution*). Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geqslant \mu_2^j \geqslant \cdots \geqslant \mu_n^j$), the solution $\mathbf{s}_j$ of the optimization problem BEST-REPLY$_j$ is given by

$$s_{ji} = \begin{cases} \frac{1}{\phi_j}\left(\mu_i^j - \sqrt{\mu_i^j}\,\frac{\sum_{i=1}^{c_j}\mu_i^j - \phi_j}{\sum_{i=1}^{c_j}\sqrt{\mu_i^j}}\right) & \text{if } 1 \leqslant i < c_j, \\ 0 & \text{if } c_j \leqslant i \leqslant n, \end{cases} \tag{8}$$

where c_j is the minimum index that satisfies the inequality

$$\sqrt{\mu_{c_j}^j} \leqslant \frac{\sum_{k=1}^{c_j}\mu_k^j - \phi_j}{\sum_{k=1}^{c_j}\sqrt{\mu_k^j}}. \tag{9}$$

Proof. In Appendix A. □

Based on the above theorem we derived the following algorithm for determining user j's best reply strategy:

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

1. Sort the computers in decreasing order of their available processing rates ($\mu_1^j \geqslant \mu_2^j \geqslant \cdots \geqslant \mu_n^j$);
2. $t \leftarrow \frac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$
3. **while** ($t \geqslant \sqrt{\mu_n^j}$) **do**
   $s_{jn} \leftarrow 0$
   $n \leftarrow n - 1$
   $t \leftarrow \frac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$
4. **for** $i = 1, \ldots, n$ **do**
   $s_{ji} \leftarrow \left(\mu_i^j - t\sqrt{\mu_i^j}\right) \frac{1}{\phi_j}$

The following theorem proves the correctness of this algorithm.

Theorem 2.2. The load balancing strategy $\{s_{j1}, s_{j2}, \ldots, s_{jn}\}$ computed by the BEST-REPLY algorithm solves the optimization problem BEST-REPLY$_j$ and is the best reply strategy of user j.

Proof. In Appendix A. □

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

Example 1. Let us apply the BEST-REPLY algorithm to a system of 3 computers and only one user. The computers have the following available processing rates: $\mu_1^1 = 10.0$, $\mu_2^1 = 2.0$ and $\mu_3^1 = 1.0$. The total arrival rate is $\phi_1 = 6.0$. The computers are already sorted in decreasing order of their processing rate and we execute Step 2 in which we compute t.

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

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

3. A distributed load balancing algorithm

The computation of Nash equilibrium may require some coordination between the users. In our case this is necessary in the sense that users need to coordinate in order to obtain the load information from each computer. From the practical point of view we need decentralization and this can be obtained by using distributed greedy best reply algorithms [3]. In these algorithms each user updates from time to time its load balancing strategy by computing the best reply against the existing load balancing strategies of the other users.

Based on the BEST-REPLY algorithm presented in the previous section, we devise the following greedy best reply algorithm for computing the Nash equilibrium for our noncooperative load balancing game. In this algorithm users are synchronized such that they update their strategies in a round-robin fashion.

We use the following notations in addition to those of Section 2:

j—the user number;
l—the iteration number;
$\mathbf{s}_j^{(l)}$—the strategy of user j computed at iteration l;
$D_j^{(l)}$—user j's expected execution time at iteration l;
ε—a properly chosen acceptance tolerance;
norm—the L^1-norm at iteration l, defined as $norm = \sum_{j=1}^{m} |D_j^{(l-1)} - D_j^{(l)}|$
Send(j, (p, q, r))—send the message (p, q, r) to user j;
Recv(j, (p, q, r))—receive the message (p, q, r) from user j;
(where p is a real number, and q, r are integer numbers).

NASH distributed load balancing algorithm:

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

1. Initialization :
   $\mathbf{s}_j^{(0)} \leftarrow \mathbf{0}$;
   $\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;
   $l \leftarrow 0$;
   $norm \leftarrow 1$;
   $sum \leftarrow 0$;
   $tag \leftarrow$ CONTINUE;
   $left = [(j - 2) \bmod m] + 1$;
   $right = [j \bmod m] + 1$;
2. **while** (1) **do**
   if($j = 1$) {user 1}
   if ($l \neq 0$)
   Recv(*left*, (*norm*, *l*, *tag*));
   if ($norm < \varepsilon$)
   Send(*right*, (*norm*, *l*, STOP));
   exit;

```
            sum ← 0;
            l ← l + 1;
        else {the other users}
            Recv(left, (sum, l, tag));
            if (tag = STOP)
                if (j ≠ m)   Send(right, (sum,
                                l, STOP));
                exit;
        for i = 1, ..., n do
            Obtain μ_i^j by inspecting the run queue
            of each computer
            (μ_i^j ← μ_i − Σ_{k=1,k≠j}^{m} s_ki φ_k);
        s_j^(l) ← BEST − REPLY (μ_1^j, ..., μ_n^j, φ_j);
        Compute D_j^(l);
        sum ← sum + |D_j^(l−1) − D_j^(l)|;
        Send(right, (sum, l, CONTINUE));
    endwhile
```

The execution of this algorithm is restarted periodically or when the system parameters are changed. Once the Nash equilibrium is reached, the users will continue to use the same strategies and the system remains in equilibrium. This equilibrium is maintained until a new execution of the algorithm is initiated.

In multiprogrammed heterogeneous distributed systems [26], the NASH scheme works as follows. Each user has an associated scheduler agent (process) which makes the allocation decisions and communicates with the scheduling agents of the other users in the system. The NASH algorithm is executed periodically by this set of scheduling agents. The scheduling agent estimates the job arrival rate at the user by considering the number of arrivals over a fixed interval of time (as presented in [2]). It also queries the state of each computer in the system and based on the estimated available processing rate reported by computers it decides the fractions s_{ij}. The estimation of available processing rate is done using the technique presented in [2]. Once the fractions are determined the scheduling agent sends the next job to computer j with probability s_{ij}. This type of allocation based on s_{ij} fractions was studied before in [35].

An important practical question is whether such 'best reply' algorithms converge to the Nash equilibrium. The only known results about the convergence of such algorithms have been obtained in the context of routing in parallel links. These studies have been limited to special cases of two parallel links shared by two users [30] or by $m \geqslant 2$ users but with linear cost links [1]. For M/M/1-type cost functions there is no known proof that such algorithms converge for more than two users. As shown by several experiments done on different settings, these algorithms may converge for more than two users. In the next section, we present such experiments that confirm this hypothesis (also see [4]). The convergence proof for more than two users is still an open problem.

4. Experimental results

4.1. Simulation environment

The simulations were carried out using Sim++ [7], a simulation software package written in C++. This package provides an application programming interface which allows the programmer to call several functions related to event scheduling, queueing, preemption and random number generation. The simulation model consists of a collection of computers connected by a communication network. Jobs arriving at the system are distributed to the computers according to the specified load balancing scheme. Jobs which have been dispatched to a particular computer are *run-to-completion* (i.e. no preemption) in first-come-first-served (FCFS) order.

Each computer is modeled as an M/M/1 queueing system [18]. The main performance metrics used in our simulations are the *expected response time* and the *fairness index*. The *fairness index*

$$I(\mathbf{D}) = \frac{[\sum_{j=1}^{m} D_j]^2}{m \sum_{j=1}^{m} D_j^2} \tag{10}$$

was proposed in [14] to quantify the fairness of load balancing schemes. Here, the parameter $\mathbf{D}$ is the vector $\mathbf{D} = (D_1, D_2, \ldots, D_m)$ where D_j is the expected execution time of user j's jobs. This index is a measure of the 'equality' of users' job execution times. If all the users have the same expected job execution times then $I = 1$ and the system is 100% fair to all users and it is load balanced. If the differences on D_j increase, I decreases and the load balancing scheme favors only some users.

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

4.2. Performance evaluation

For comparison purposes we consider three existing static load balancing schemes [6,15,16]. A brief description of these schemes is given below:

- *Proportional scheme (PS)* [6]. According to this scheme each user allocates its jobs to computers in proportion to their processing rate. This allocation seems to be a natural choice but it may not minimize the user's expected response time or the overall expected response time. The fairness index for this scheme is always 1 as can be easily seen from Eq. (10).
- *Global optimal scheme (GOS)* [16]. This scheme minimizes the expected execution time over all jobs executed by the system. The load fractions (**s**) are obtained by

solving the following nonlinear optimization problem:

$$\min_{\mathbf{s}} \frac{1}{\Phi} \sum_{k=1}^{m} \phi_j D_j(\mathbf{s}) \tag{11}$$

subject to the constraints

$$s_{ji} \geqslant 0, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m, \tag{12}$$

$$\sum_{i=1}^{n} s_{ji} = 1, \quad j = 1, \ldots, m, \tag{13}$$

$$\sum_{j=1}^{m} s_{ji} \phi_j < \mu_i, \quad i = 1, \ldots, n. \tag{14}$$

This scheme provides the overall optimum for the expected execution time but it is not user-optimal and is unfair.

- *Individual optimal scheme* (*IOS*) [15]. In this scheme, each job optimizes its response time for itself independently of others. In general, the Wardrop equilibrium, which is the solution given by this scheme, is not optimal and in some cases we expect worse response time than the other policies [15]. It is based on an iterative procedure that is not very efficient. For a complete description of IOS algorithm see [15]. The advantage of this scheme is that it provides a fair allocation.

Remark. Among the schemes described above, the IOS scheme is the only scheme that is based on game theoretic concepts. Our scheme (NASH) and PS are the only distributed schemes considered in this paper. IOS and GOS are centralized schemes.

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

4.2.1. The convergence of NASH algorithm

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

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

1. Initialization:
   for $i = 1, \ldots, n$ **do**
   $s_{ji}^{(0)} \leftarrow \frac{\mu_i}{\sum_{k=1}^{n} \mu_k}$;
   $\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;
   $l \leftarrow 0$;
   ⋮

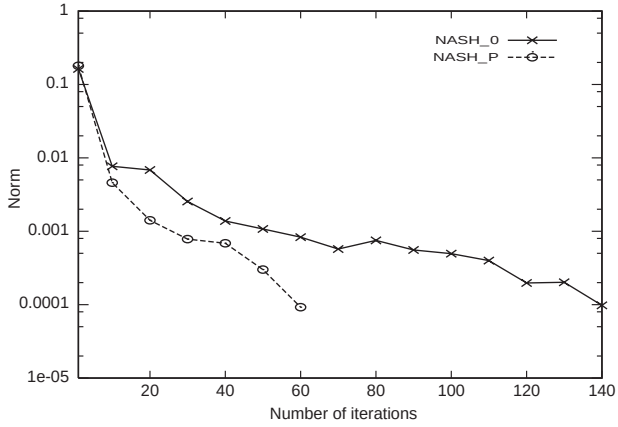

Fig. 2. Norm vs. number of iterations.

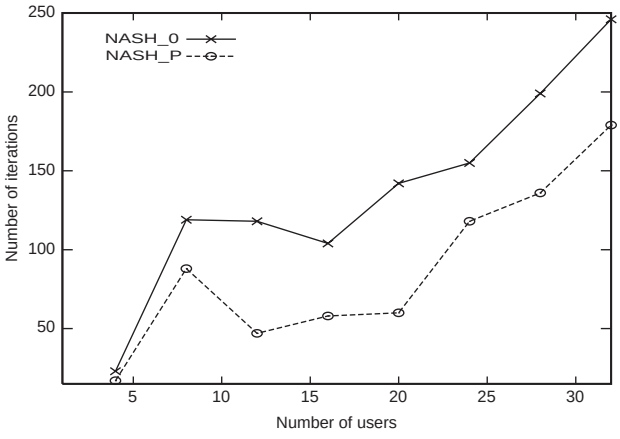

Fig. 3. Convergence of best reply algorithms (until $norm < 10^{-4}$).

We call this new version NASH_P. Using this initialization the starting point will be a proportional allocation of jobs to computers according to their processing rate. We expect a better convergence using NASH_P instead of NASH_0. To study the convergence of these algorithms we consider a system with 16 computers shared by 10 users. The norm vs. the number of iterations is shown in Fig. 2. It can be seen that the NASH_P algorithm significantly outperforms NASH_0 algorithm. The intuitive explanation for this performance is that the initial proportional allocation is close to the equilibrium point and the number of iterations needed to reach the equilibrium is reduced. Using the NASH_P algorithm the number of iterations needed to reach the equilibrium is reduced by more than a half compared with NASH_0.

Next, we study the influence of the number of users on the convergence of both algorithms. In Fig. 3, we present the number of iterations needed to reach the equilibrium ($norm < 10^{-4}$) for a system with 16 computers and a variable number of users (from 4 to 32). It can be observed that NASH_P significantly outperforms NASH_0 reducing the number of iterations needed to reach the equilibrium in all the cases.

Table 1
System configuration

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

Table 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

4.2.2. *Effect of system utilization*

To study the effect of system utilization we simulated a heterogeneous system consisting of 16 computers with four different processing rates. This system is shared by 10 users. In Table 1, we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We consider only computers that are at most 10 times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

For each experiment 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

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i}. \tag{15}$$

We choose fixed values for the system utilization and we determined the total job arrival Φ. For example, if we consider $\rho = 10\%$ and an aggregate processing rate of 510 jobs/s then the arrival rate in the system is $\lambda = 51$ jobs/s. The job arrival rate for each user ϕ_j, $j = 1, \ldots, 10$ is determined from the total arrival rate as $\phi_j = q_j \Phi$, where the fractions q_j are given in Table 2.

In Fig. 4, we present the expected response time of the system and the fairness index for different values of system utilization (ranging from 10% to 90%). It can be observed that at low loads (ρ from 10% to 40%) all the schemes except PS yield almost the same performance. The poor performance of PS scheme is due to the fact that the less powerful computers are significantly overloaded.

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

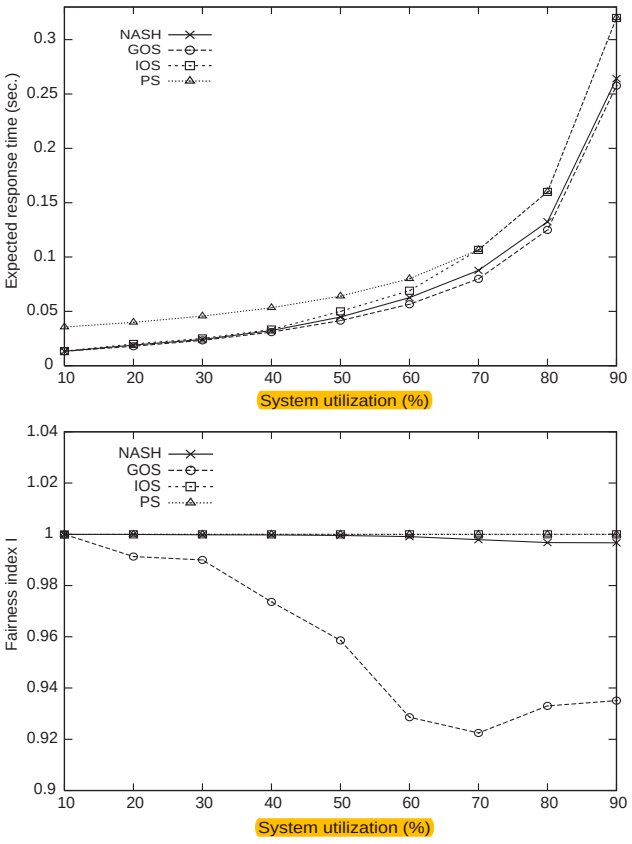

Fig. 4. The expected response time and fairness index vs. system utilization.

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

The PS and IOS schemes maintain a fairness index of 1 over the whole range of system loads. It can be shown that the PS has a fairness index of 1 which is a constant independent of the system load. The fairness index of GOS varies from 1 at low load, to 0.92 at high load. The NASH scheme has a fairness index close to 1 and each user obtains the minimum possible expected response time for its own jobs given what every other user is doing (i.e. it is user-optimal). The stability of the allocation under noncooperative behavior and decentralization are the main advantages of NASH scheme.

An interesting issue is the impact of static load balancing schemes on individual users. In Fig. 5, we present the expected response time for each user considering all static schemes at medium load (ρ=60%). The PS and IOS schemes guarantee equal expected response times for all users but with the disadvantage of a higher expected execution time for their jobs. It can be observed that in the case of GOS scheme there are large differences in users' expected exe-

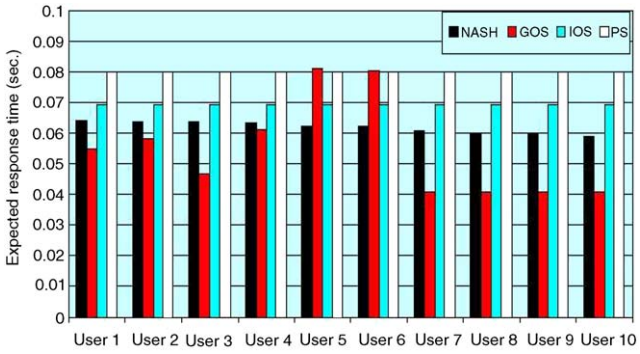

Fig. 5. Expected response time for each user.

cution times. NASH scheme provides the minimum possible expected execution time for each user (according to the properties of the Nash equilibrium).

4.2.3. *Effect of heterogeneity*

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

In this section, we investigate the effectiveness of load balancing schemes by varying the speed skewness. We simulate a system of 16 heterogeneous computers: 2 fast and 14 slow. The slow computers have a relative processing rate of 1 and we varied the relative processing rate of the fast computers from 1 (which corresponds to a homogeneous system) to 20 (which corresponds to a highly heterogeneous system). The system utilization was kept constant $\rho = 60\%$. The fractions used to determine the job arrival rate of each user are those presented in Table 2.

In Fig. 6, we present the effect of speed skewness on the expected response time and fairness. It can be observed that increasing the speed skewness the GOS and NASH schemes yield almost the same expected response time which means that in highly heterogeneous systems the NASH scheme is very effective. NASH scheme has the additional advantage of decentralization and user-optimality which is very important in actual distributed systems. PS scheme performs poorly because it overloads the slowest computers. The IOS scheme performs well at high-speed skewness approaching the performance of NASH and GOS, but at low-speed skewness it performs poorly.

At low-speed skewness IOS obtains the same expected response time as PS. This is due to the fact that at low-speed skewness IOS determines the same allocation as PS, overloading the slow computers. When the processing rate of the

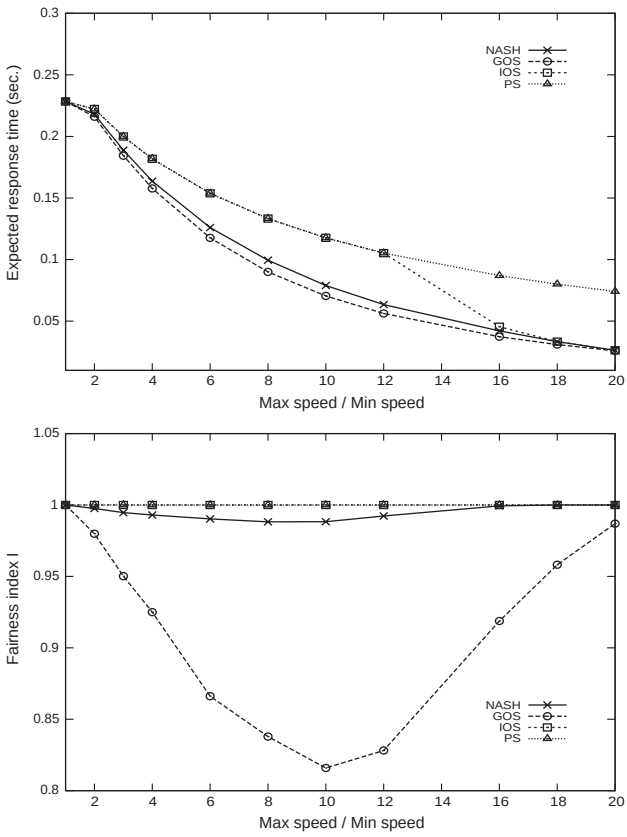

Fig. 6. The effect of heterogeneity on the expected response time and fairness index.

fastest computer is increased more than 16 times IOS is able to determine a better allocation than that of PS, allocating fewer jobs to the slow computers.

The fairness index of NASH is very close to one, which is the value obtained by PS and IOS. These three schemes guarantee equal expected response times for all the users in the system over all range of speed skewness. The fairness index of GOS is between 0.95 and 1 at low- and high-speed skewness and between 0.8 and 0.85 at medium speed skewness. These results show that at medium speed skewness GOS scheme produces an allocation which does not guarantee equal expected response times for all the users in the system.

Remark. In homogeneous systems all the schemes considered in this paper obtain the same expected response time and the same fairness index. This is because all computers in the system have the same processing rate and all the load balancing schemes presented here will allocate an equal amount of load to each computer.

4.2.4. *Effect of system size*

An important issue is to study the influence of system size on the performance of load balancing schemes. To study this issue we performed two types of experiments on a simulated heterogeneous distributed system consisting of two types of

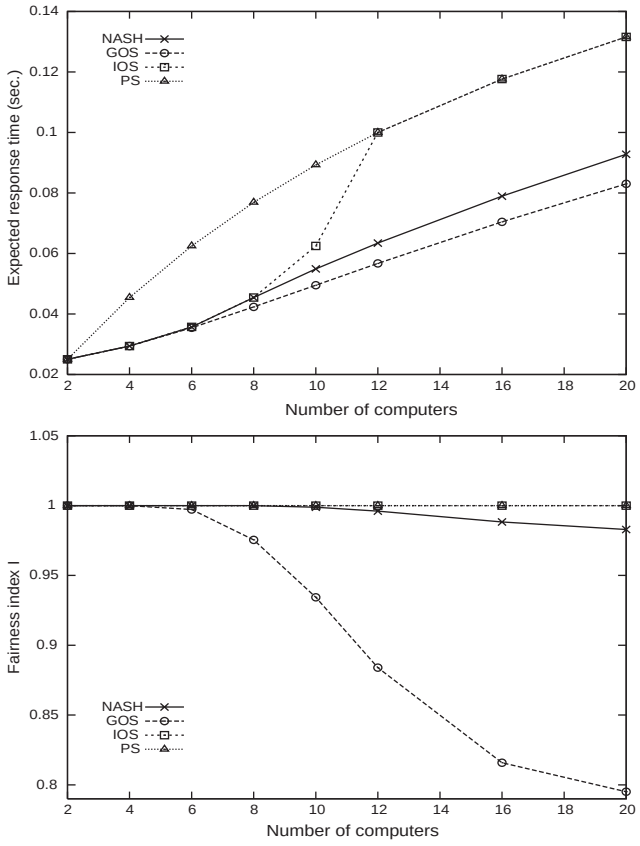

Fig. 7. The effect of system size on the expected response time and fairness index.

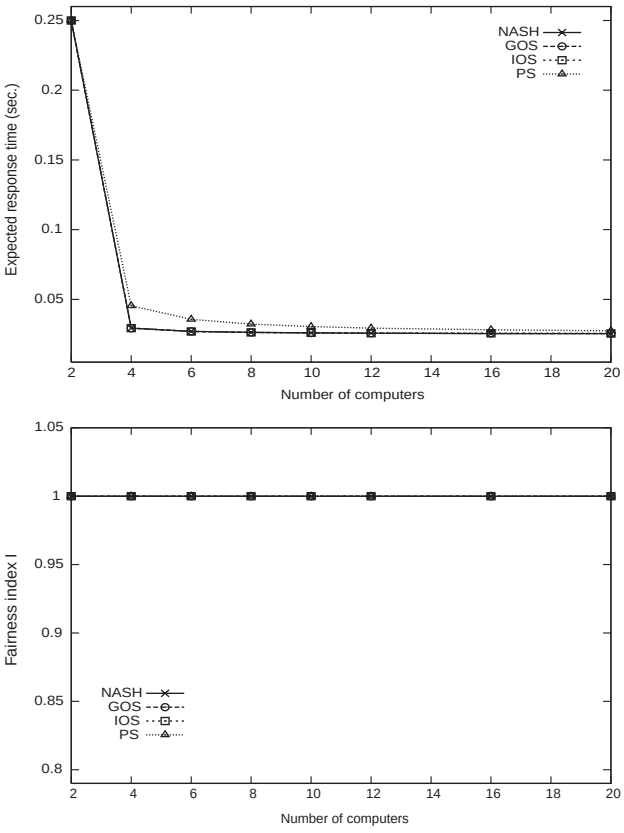

Fig. 8. The effect of system size on the expected response time and fairness index.

computers: slow computers (relative processing rate = 1) and fast computers (relative processing rate = 10). In all these experiments the system utilization was kept constant, $\rho = 60\%$.

In the first type of experiments we gradually increase the size of the system by adding slow computers, more precisely we increase the number of computers from 2 (fast computers only) to 20 (2 fast and 18 slow computers). Fig. 7 shows the expected response time and the fairness index for all the schemes. The performance of NASH and GOS is almost the same when we have few computers (2–8). The PS scheme performs poorly even for a small system. The expected response time for IOS degrades increasing the system size and approaches the expected response time of PS. This is because when we add slow computers IOS allocates more jobs to the slow computers in the system overloading them. NASH guarantees a good performance for medium and large systems and the same performance as GOS for small systems. The additional advantage of NASH is that it is a distributed scheme providing a user-optimal allocation. The fairness index obtained by NASH is 1 when we have a small system and between 1 and 0.98 when the size of the system is between 8 and 20. This means that NASH allocates jobs to computers such that each user will obtain the same expected execution time. As seen from the results GOS does not guarantee equal expected execution times for all the users when the system size increases to more than 8 computers.

In the second type of experiments we gradually increase the size of the system by adding fast computers, more precisely we increase the number of computers from 2 (slow computers only) to 20 (2 slow and 18 fast computers). Fig. 8 shows the expected response time and the fairness index for all the schemes. The performance of NASH, GOS and IOS is the same. This is because adding fast computers in the system makes these schemes allocate no jobs to the two slow computers. Thus, only the fast computers having the same speed receive jobs and the system can be considered as a homogeneous system. In this case all the schemes except PS will obtain similar performance. PS does not obtain the optimum because it allocates some jobs to the two slow computers increasing the overall expected execution time. The high value of the expected execution time when the system size is two is due to the fact that we have only the slow computers in the system and they are overloaded. As seen from the results the fairness index is the same for all the schemes over all range of system sizes. As explained above this is because the system behaves similarly to a homogeneous system. NASH provides the additional advantage of being a distributed load balancing scheme.

5. Conclusion

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

Future work will address the development of game theoretic models for load balancing in the context of uncertainty as well as game theoretic models for dynamic load balancing.

Acknowledgments

The authors express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of the paper.

This research was supported, in part, by research grants from: NSF CCR-0312323, NASA NAG 2-1383 (1999-2001), and State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999.

Appendix A

In this section, we present the proofs of the results used in the paper.

Proof of Theorem 2.1. We begin with the observation that at the Nash equilibrium the stability condition (7) is always satisfied because of (3) and the fact that the total arrival rate (Φ) does not exceed the total processing rate of the distributed system. Thus we consider OP_j problem with only two restrictions, (5) and (6).

We first show that $D_j(\mathbf{s})$ is a convex function in $\mathbf{s}_j$ and that the set of feasible solutions defined by the constraints (5) and (6) is convex.

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

Thus, OP_j involves minimizing a convex function over a convex feasible region and the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality [23].

Let $\alpha \geqslant 0$, $\eta_i \geqslant 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [23]. The Lagrangian is

$$L(s_{j1}, \ldots, s_{jn}, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i^j - s_{ji}\phi_j} - \alpha\left(\sum_{i=1}^{n} s_{ji} - 1\right) - \sum_{i=1}^{n} \eta_i s_{ji}. \tag{16}$$

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

$$\frac{\partial L}{\partial s_{ji}} = 0, \tag{17}$$

$$\frac{\partial L}{\partial \alpha} = 0, \tag{18}$$

$$\eta_i s_{ji} = 0, \quad \eta_i \geqslant 0, \quad s_{ji} \geqslant 0, \quad i = 1, \ldots, n. \tag{19}$$

These conditions become

$$\frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2} - \alpha - \eta_i = 0, \quad i = 1, \ldots, n, \tag{20}$$

$$\sum_{i=1}^{n} s_{ji} = 1, \tag{21}$$

$$\eta_i s_{ji} = 0, \quad \eta_i \geqslant 0, \quad s_{ji} \geqslant 0, \quad i = 1, \ldots, n. \tag{22}$$

These are equivalent to

$$\alpha = \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if } s_{ji} > 0, \quad 1 \leqslant i \leqslant n, \tag{23}$$

$$\alpha \leqslant \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if } s_{ji} = 0, \quad 1 \leqslant i \leqslant n, \tag{24}$$

$$\sum_{i=1}^{n} s_{ji} = 1, \quad s_{ji} \geqslant 0, \quad i = 1, \ldots, n. \tag{25}$$

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

From (23) and based on the above claims we can obtain by summation the following equation:

$$\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}=\sqrt{\alpha}\left(\sum_{i=1}^{c_j-1}\mu_i^j-\sum_{i=1}^{c_j-1}s_{ji}\phi_j\right). \tag{26}$$

Using (24) the above equation becomes

$$\sqrt{\alpha}=\frac{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1}\mu_i^j-\sum_{i=1}^{c_j-1}s_{ji}\phi_j}\leqslant\frac{1}{\sqrt{\mu_i^{c_j}}}. \tag{27}$$

This is equivalent to

$$\sqrt{\mu_i^{c_j}}\sum_{i=1}^{c_j}\sqrt{\mu_i^j}\leqslant\sum_{i=1}^{c_j}\mu_i^j-\phi_j. \tag{28}$$

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

Proof of Theorem 2.2. The while loop in step 3 finds the minimum index c_j for which $\sqrt{\mu_{c_j}^j}\leqslant\frac{\sum_{k=1}^{c_j}\mu_k^j-\phi_j}{\sum_{k=1}^{c_j}\sqrt{\mu_k^j}}$. In the same loop, s_{ji} are set to zero for $i=c_j,\ldots,n$. In step 4, s_{ji} is set equal to $\frac{1}{\phi_j}\left(\mu_i^j-\sqrt{\mu_i^j}\frac{\sum_{i=1}^{c_j}\mu_i^j-\phi_j}{\sum_{i=1}^{c_j}\sqrt{\mu_i^j}}\right)$ for $i=1,\ldots,c_j-1$. These are in accordance with Theorem 2.1. Thus, the allocation $\{s_{j1},\ldots,s_{jn}\}$ computed by the BEST-REPLY algorithm is the optimal solution of BEST-REPLY$_j$. □

References

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

[2] L. Anand, D. Ghose, V. Mani, ELISA: an estimated load information scheduling algorithm for distributed computing systems, Comput. Math. Appl. 37 (1999) 57–85.

[3] T. Basar, G.J. Olsder, Dynamic Noncooperative Game Theory, SIAM, Philadelphia, PA, 1998.

[4] T. Boulogne, E. Altman, O. Pourtallier, On the convergence to Nash equilibrium in problems of distributed computing, Ann. Oper. Res. 109 (1) (January 2002) 279–291.

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

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

[7] R.M. Cubert, P. Fishwick, Sim++ Reference Manual, CISE, University of Florida, July 1995.

[8] A.A. Economides, J. Silvester, A game theory approach to cooperative and non-cooperative routing problems, in: ITS '90, Proceedings of the Telecommunication Symposium 1990, pp. 597–601.

[9] A.A. Economides, J. Silvester, Multi-objective routing in integrated services networks: a game theory approach, in: INFOCOM '91, Proceedings of the 10th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, April 1991, pp. 1220–1227.

[10] R. Freund, H.J. Siegel, Heterogeneous processing, IEEE Comput. Mag. 26 (6) (June 1993) 13–17.

[11] D. Fudenberg, J. Tirole, Game Theory, The MIT Press, Cambridge, MA, 1994.

[12] A. Ghafoor, J. Yang, A distributed heterogeneous supercomputing management system, IEEE Comput. Mag. 26 (6) (June 1993) 78–86.

[13] D. Grosu, A.T. Chronopoulos, M.Y. Leung, Load balancing in distributed systems: an approach using cooperative games, in: Proceedings of the International Parallel and Distributed Processing Symposium, April 2002, pp. 52–61.

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

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

[16] C. Kim, H. Kameda, Optimal static load balancing of multi-class jobs in a distributed computer system, in: Proceedings of the 10th International Conference on Distributed Computing Systems, May 1990, pp. 562–569.

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

[18] L. Kleinrock, Queueing Systems—vol. 1: Theory, Wiley, New York, 1975.

[19] Y.A. Korilis, A.A. Lazar, A. Orda, Capacity allocation under noncooperative routing, IEEE Trans. Automat. Control 42 (3) (March 1997) 309–325.

[20] E. Koutsoupias, C. Papadimitriou, Worst-case equilibria, in: Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, 1999, pp. 404–413.

[21] H. Lee, Optimal static distribution of prioritized customers to heterogeneous parallel servers, Comput. Oper. Res. 22 (10) (December 1995) 995–1003.

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

[23] D.G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, Reading, MA, 1984.

[24] M. Mavronicolas, P. Spirakis, The price of selfish routing, in: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, July 2001, pp. 510–519.

[25] R. Mirchandaney, D. Towsley, J. Stankovic, Adaptive load sharing in heterogeneous systems, in: Proceedings of the Ninth IEEE International Conference on Distributed Computing Systems, June 1989, pp. 298–306.

[26] V.K. Naik, S.K. Setia, M.S. Squillante, Processor allocation in multiprogrammed distributed memory parallel computer systems, J. Parallel Distributed Comput. 46 (1997) 28–47.

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

[28] J. Nash, Non-cooperative games, Ann. Math. 54 (2) (September 1951) 286–295.

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

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

[31] M. Osborne, An Introduction to Game Theory, Oxford University Press, New York, 2004.

[32] K.W. Ross, D.D. Yao, Optimal load balancing and scheduling in a distributed computer system, J. Assoc. Comput. Mach. 38 (3) (July 1991) 676–690.

[33] T. Roughgarden, Stackelberg scheduling strategies, in: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, July 2001, pp. 104–113.

[34] T. Roughgarden, E. Tardos, How bad is selfish routing?, in: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science, November 2000, pp. 93–102.
[35] X. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: Proceedings of the International Conference on Parallel Processing, August 2000, 373–382.
[36] A.N. Tantawi, D. Towsley, Optimal static load balancing in distributed computer systems, J. Assoc. Comput. Mach. 32 (2) (April 1985) 445–465.
[37] M.H. Willebeek-LeMair, A.P. Reeves, Strategies for dynamic load balancing on highly parallel computers, IEEE Trans. Parallel Distributed Systems 4 (9) (September 1993) 979–993.
[38] J. Yang, I. Ahmad, A. Ghafoor, Estimation of execution times on heterogeneous supercomputing architectures, in: Proceedings of the International Conference on Parallel Processing, 1993, pp. 219–226.

Daniel Grosu received his Diploma in Engineering (Automatic Control and Industrial Informatics) from the Technical University of Iasi, Romania in 1994 and the M.Sc. and Ph.D. degrees in Computer Science from The University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an Assistant Professor in the Department of Computer Science at Wayne State University, Detroit. His research interests include load balancing, distributed systems, electronic voting, security and topics at the border of computer science, game theory and economics. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a member of the IEEE, ACM and the SIGACT.

Anthony T. Chronopoulos received his Ph.D. at the University of Illinois in Urbana-Champaign in 1987. He is a senior member of the IEEE and the ACM. He has published 36 journal and 45 refereed conference proceedings publications in the areas of distributed systems, game theory, networks and security, parallel processing. He has been awarded 12 federal/state government research grants. His work is cited in more than 190 nonco-authors' research articles. He has advised three Ph.D. students who are active researchers.