CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2008)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1331

Cooperative load balancing in distributed systems

D. Grosu[1],*,†, A. T. Chronopoulos[2] and M. Y. Leung[3]

[1]*Department of Computer Science*, *Wayne State University*, *Detroit*, *MI 48202*, *U.S.A.*
[2]*Department of Computer Science*, *The University of Texas at San Antonio*, *San Antonio*, *TX 78249*, *U.S.A.*
[3]*Department of Mathematical Sciences*, *University of Texas at El Paso*, *El Paso*, *TX 79968*, *U.S.A.*

SUMMARY

A serious difficulty in concurrent programming of a distributed system is how to deal with scheduling and load balancing of such a system which may consist of heterogeneous computers. In this paper, we formulate the static load-balancing problem in single class job distributed systems as a cooperative game among computers. The computers comprising the distributed system are modeled as M/M/1 queueing systems. It is shown that the Nash bargaining solution (NBS) provides an optimal solution (operation point) for the distributed system and it is also a fair solution. We propose a cooperative load-balancing game and present the structure of NBS. For this game an algorithm for computing NBS is derived. We show that the fairness index is always equal to 1 using NBS, which means that the solution is fair to all jobs. Finally, the performance of our cooperative load-balancing scheme is compared with that of other existing schemes. Copyright © 2008 John Wiley & Sons, Ltd.

Received 16 September 2007; Revised 14 January 2008; Accepted 10 February 2008

KEY WORDS: distributed systems; static load balancing; game theory; Nash bargaining solution; cooperative game

1. INTRODUCTION

A distributed system can be viewed as a collection of computing and communication resources shared by active users. Often, jobs in a distributed system can be divided into different classes

*Correspondence to: D. Grosu, Department of Computer Science, Wayne State University, 5143 Cass Avenue, Detroit, MI 48202, U.S.A.
†E-mail: dgrosu@cs.wayne.edu

Contract/grant sponsor: NASA; contract/grant number: NAG 2-1383
Contract/grant sponsor: NSF; contract/grant numbers: ASC-9634775, CCR-0312323, DGE-0654014

based on their resource usage characteristics and ownership. For example, the jobs that belong to a single user can form a class. Alternatively, we can distinguish different classes of jobs by their execution times. Depending on how many job classes are considered we can have single class or multi-class job distributed systems. In this paper, we consider the load-balancing problem in single class job distributed systems. The single class job system refers to jobs that have the same computational requirements and take unit time to execute.

There are three typical approaches to load-balancing problem in single class job distributed systems: global, non-cooperative and cooperative.

In the *global approach* there exists only one decision maker that optimizes the response time of the entire system over all jobs. The goal is to obtain a system-wide optimal allocation of jobs to computers. This approach has been studied extensively using techniques such as nonlinear optimization [1,2] and polymatroid optimization [3]. The load-balancing schemes that implement the global approach are centralized and determine a load allocation resulting in a system-wide optimal response time.

In the *non-cooperative approach* each of (infinitely or finitely) many jobs optimizes its own response time independently of the others and they all eventually reach an equilibrium. This situation can be modeled as a non-cooperative game among jobs. At the equilibrium solution a job cannot receive any further benefit by changing its own decision. For an infinite number of jobs, this problem has been studied in [4]. The equilibrium is called *Wardrop equilibrium*. For a finite number of jobs, the equilibrium is the *Nash equilibrium* [5]. The non-cooperative load-balancing schemes are distributed. Their main drawback is that under certain conditions [6] the equilibrium load allocation provides a suboptimal system-wide response time.

In the *cooperative approach* several decision makers (e.g. jobs, computers) cooperate in making the decisions so that each of them will operate at its optimum. The 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 [5]. The cooperative schemes provide a Pareto-optimal response time and also guarantee the fairness of the resource allocation to all the jobs. The fairness of allocation considered here requires that the jobs obtain the same response time independent of the allocated computer.

Related work: The problem of static load balancing in single class job distributed systems has been studied in the past using the global and the non-cooperative approach. This paper investigates the cooperative approach. Preliminary results on this were reported in our previous paper [7].

The focus of the *global approach* is on minimizing the expected response time of the entire system over all jobs. Tantawi and Towsley [2] formulated the load-balancing problem as a nonlinear optimization problem and gave an algorithm for computing the allocation. Kim and Kameda [8] derived a more efficient algorithm to compute the allocation. Li and Kameda [9,10] proposed algorithms for static load balancing in star and tree networks. Tang and Chanson [1] proposed and studied several static load-balancing schemes that take into account the job dispatching strategy. Also, there exist several studies on static load balancing in multi-class job systems [11–14]. A closely related problem to the static load-balancing problem studied in this paper is the scheduling of divisible loads. Scheduling divisible loads problem is the subject of divisible load theory (DLT) [15]. DLT considers the scheduling of arbitrarily divisible loads on a distributed computing system. DLT problems have large data sets where every element within the set requires an identical type of processing. The set can be partitioned into any number of fractions where each fraction requires

DOI: 10.1002/cpe

scheduling. These problems commonly arise in many domains including image processing [16], databases [17], linear algebra [18], visualization [19] and multimedia broadcasting [20]. Several divisible load scheduling algorithms based on the global approach were proposed in [15].

There exist only few studies that use the *non-cooperative approach* for load balancing in distributed systems. A load-balancing algorithm for single class job systems that computes the Wardrop equilibrium was proposed by Kameda *et al.* [4]. They also proposed a load-balancing algorithm for multi-class job systems based on the non-cooperative approach. Grosu and Chronopoulos [21] proposed a game-theoretic model and an algorithm for load balancing based on Nash equilibrium. Penmatsa and Chronopoulos [22] extended the non-cooperative model proposed in [21] to include resource pricing. Roughgarden [23] formulated the load-balancing problem as a Stackelberg game and showed that it is NP-hard to compute the optimal Stackelberg allocation strategy. The problem of routing traffic in networks assuming non-cooperative game models was studied in [24–28]. Several researchers considered the design of algorithms for resource allocation involving self-interested participants [29–31]. Nisan and Ronen [31] proposed mechanisms that solve the problem of task scheduling on selfish unrelated machines.

We are not aware of any research employing the *cooperative approach* to solve the static load-balancing problem in distributed systems. The cooperative approach was used extensively in networking to solve the problem of fair bandwidth allocation between applications [32–34].

Motivation and contribution: The existing studies on static load balancing considered optimal allocations that minimize the overall system expected response time. However, in such allocations some jobs may experience much longer expected response time than others. We consider the problem of static load balancing that has the objective of finding an allocation of jobs to computers that yields an equal or approximately equal expected response time for all the jobs. We say that such an allocation is fair (to all the jobs). Fairness of allocation is a major issue in modern utility computing systems such as Amazon Elastic Compute Cloud [35] and Sun Grid Compute Utility [36]. In such systems, users pay the same price for the compute capacity they actually consume. Guaranteeing the fairness of allocation to the users in such fixed price settings is an important and difficult problem. Our goal is to find a formal framework for characterizing and designing fair allocation schemes. The framework is provided by cooperative game theory. Using this framework we formulate the load-balancing problem in single class job distributed systems as a cooperative game among computers. We show that the *Nash Bargaining Solution* (NBS) provides a Pareto-optimal operation point for the distributed system. We give a characterization of NBS and an algorithm for computing it. We prove Pareto optimality and fairness of the allocation. We study the performance of our proposed load-balancing scheme via simulation. We make comparisons with two other static representative schemes: (i) a scheme that yields the system-wide optimal expected response time that is used as a baseline scheme for our experiments and (ii) a scheme that allocates jobs to computers in proportion to their computing power and yields the worst expected response time of the static schemes in the literature. This comparison shows that our new allocation scheme yields a very good performance and fairness. To the best of our knowledge this is the first work that models the static load-balancing problem in distributed systems as a cooperative game that uses the NBS as a solution. Previous research considered the cooperative approach and the NBS for solving the bandwidth allocation problem in networking [34].

Organization: This paper is structured as follows. In Section 2 we present the NBS concept for cooperative games and state several lemmas and theorems needed for our results. In Section 3 we

DOI: 10.1002/cpe

introduce our cooperative load-balancing game and derive an algorithm for computing the solution. In Section 4 the performance and fairness of our cooperative solution is compared with those of other existing solutions.

2. COOPERATIVE GAME THEORY CONCEPTS

In a cooperative game, players have complete freedom of preplay communication to make joint agreements about their operating points. A formal definition of a cooperative game is as follows.

Definition 2.1 (*A cooperative game*). A cooperative game consists of

(i) *M players*.

(ii) A non-empty, closed and convex set $X \subseteq \mathbf{R}^M$, which is the *set of strategies* of the M players.

(iii) For each player i, $i = 1, 2, \ldots, M$, an *objective function* is f_i. Each f_i is a function from X to $\mathbf{R}$ and it is bounded above. The goal is to maximize simultaneously all $f_i(x)$.

(iv) For each player i, $i = 1, 2, \ldots, M$, a *minimal value* of f_i, denoted u_i^0, is required by player i without any cooperation to enter the game. The vector $\mathbf{u}^0 = (u_1^0, u_2^0, \ldots, u_M^0)$ is called the initial *agreement point*.

Example 1 (*A cooperative game in distributed systems*). In the context of resource allocation in distributed systems the players could represent computers. The objective function $f_i(x)$ could represent the inverse of the expected response time at computer i, where x is the vector of job arrival rates at each computer. The initial values u_i^0 of each computer represent a minimum performance guarantee the computers must provide to the users.

Definition 2.2 (*The set of achievable performances*). The *set of achievable performances with respect to the initial agreement point* $\mathbf{u}^0$ is the set G defined by $G = \{(U, \mathbf{u}^0) | U \subset \mathbf{R}^M$ is a non-empty convex and bounded set, and $\mathbf{u}^0 \in \mathbf{R}^M$ is such that $U_0 = \{\mathbf{u} \in U | \mathbf{u} \geq \mathbf{u}^0\}$ is non-empty$\}$ where $U \subset \mathbf{R}^M$ is the set of achievable performances.

We are interested in finding solutions for the cooperative game defined above that are Pareto optimal. In the following we define the notion of Pareto optimality [37].

Definition 2.3 (*Pareto optimality*). Assume that $x \in X$ and $\mathbf{f}(x) = (f_1(x), \ldots, f_M(x))$, $\mathbf{f}(x) \in U$. Then x is said to be *Pareto optimal* if for each $x' \in X$, $f_i(x') \geq f_i(x)$, $i = 1, \ldots, M$ imply $f_i(x') = f_i(x)$, $i = 1, \ldots, M$.

Remark. Pareto optimality means that it is impossible to find another point that leads to superior performance for one player, without strictly decreasing the performance of another player. For the game given in Example 1, an allocation x is Pareto optimal if it is impossible to find another allocation x' that leads to a lower response time for all the computers in the system.

In general, for an M-player game the set of Pareto-optimal points form an $M - 1$ dimensional hypersurface consisting of an infinite number of points [38]. What is the desired operating point for our system among them? To answer this question we need additional criteria for selecting the optimal point. Such criteria are the so-called fairness axioms that characterize the NBS [39]. Owing to its fairness properties the NBS is a natural candidate for the design of fair allocation schemes in distributed computing. In the following we discuss the NBS for cooperative games [40,41].

Definition 2.4 (*NBS (Stefanescu and Stefanescu* [*41*])). A mapping $S : G \to \mathbf{R}^M$ is said to be an NBS if:

(i) $S(U, \mathbf{u}^0) \in U_0$;
(ii) $S(U, \mathbf{u}^0)$ is Pareto optimal; and satisfies the following axioms:
(iii) *Linearity axiom*: If $\phi : \mathbf{R}^M \to \mathbf{R}^M$, $\phi(\mathbf{u}) = \mathbf{u}'$ with $u'_j = a_j u_j + b_j$, $a_j > 0$, $j = 1, \ldots, k$ then $S(\phi(U), \phi(\mathbf{u}^0)) = \phi(S(U, \mathbf{u}^0))$.
(iv) *Irrelevant alternatives axiom*: If $U \subset U'$, $(U, \mathbf{u}^0) \in G$ and $S(U', \mathbf{u}^0) \in U$, then $S(U, \mathbf{u}^0) = S(U', \mathbf{u}^0)$.
(v) *Symmetry axiom*: If U is symmetrical with respect to a subset $J \subseteq \{1, \ldots, M\}$ of indices (i.e. if $\mathbf{u} \in U$ and $i, j \in J, i < j$ imply $(u_1, \ldots, u_{i-1}, u_j, u_{j+1}, \ldots, u_{j-1}, u_i, u_{j+1}, \ldots, u_M) \in U$) and if $u_i^0 = u_j^0$, $i, j \in J$ then $S(U, \mathbf{u}^0)_i = S(U, \mathbf{u}^0)_j$, for $i, j \in J$.

Definition 2.5 (*Bargaining point (Stefanescu and Stefanescu* [*41*])). $\mathbf{u}^*$ is a *bargaining point* if it is given by $S(U, \mathbf{u}^0)$. We call $\mathbf{f}^{-1}(\mathbf{u}^*)$ the set of *bargaining solutions*.

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

Stefanescu and Stefanescu [41] gave the following characterization of the Nash bargaining point.

Theorem 2.1 (Nash bargaining point characterization (Stefanescu and Stefanescu [41]))**.** *Let X be a convex compact subset of $\mathbf{R}^M$. Let $f_i : X \to \mathbf{R}$, $i = 1, \ldots, M$ be concave functions, bounded above. Let $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x}))$, $U = \{\mathbf{u} \in \mathbf{R}^M | \exists \mathbf{x} \in X, \mathbf{f}(\mathbf{x}) \geq \mathbf{u}\}$, $X(\mathbf{u}) = \{\mathbf{x} \in X | \mathbf{f}(\mathbf{x}) \geq \mathbf{u}\}$ and $X_0 = X(\mathbf{u}^0)$ be the set of strategies that enable the players to achieve at least their initial performances. Then*

(i) *there exists a bargaining solution and a unique bargaining point $\mathbf{u}^*$;*
(ii) *the set of the bargaining solutions $\mathbf{f}^{-1}(\mathbf{u}^*)$ is determined as follows: Let J be the set of players able to achieve a performance strictly superior to their initial performance, i.e. $J = \{j \in \{1, \ldots, M\} | \exists \mathbf{x} \in X_0, f_j(\mathbf{x}) > u_j^0\}$. Each vector $\mathbf{x}$ in the bargaining solution set verifies $f_j(\mathbf{x}) > u_j^0$, for all $j \in J$ and solves the following maximization problem:*

$$\max_{\mathbf{x}} \prod_{j \in J} (f_j(\mathbf{x}) - u_j^0), \quad \mathbf{x} \in X_0 \tag{1}$$

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

Remark. From the assumption on J, the product in Equation (1) is positive. The players outside of J are not considered in the optimization problem.

DOI: 10.1002/cpe

Yaiche *et al*. [34] formulated an equivalent optimization problem and proved the following results. These results are needed for proving Theorems 3.1 and 3.2.

Proposition 2.1 (Yaiche *et al.* [34])**.** *If each* $f_j : X \to \mathbf{R}$, $(j \in J)$ *is one-to-one on* $X_0 \subset X$ *then* $\mathbf{f}^{-1}(\mathbf{u}^*)$ *is a singleton.*

Remark. Following the terminology used in [34], we call the point in this set the NBS in the remainder of this paper.

Theorem 2.2 (Yaiche *et al.* [34])**.** *Suppose that for each* $j \in J$, $f_j : X \to \mathbf{R}$, *is one-to-one on* X_0. *Under the assumption of Theorem* 2.1, *we consider the following problems*:

$$(P_J)\colon\ \max_{\mathbf{x}} \prod_{j\in J} (f_j(\mathbf{x}) - u_j^0), \quad \mathbf{x} \in X_0 \tag{2}$$

$$(P_J')\colon\ \max_{\mathbf{x}} \sum_{j\in J} \ln(f_j(\mathbf{x}) - u_j^0), \quad \mathbf{x} \in X_0 \tag{3}$$

then

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

The theorem above was given by Yaiche *et al.* [34] who used it to solve a cooperative game for bandwidth allocation and pricing in broadband networks. In the following section we use this theorem to derive a solution for our cooperative load-balancing game.

3. LOAD BALANCING AS A COOPERATIVE GAME AMONG COMPUTERS

We consider a single class job distributed system that consists of n heterogeneous computers. Here the single class job system refers to jobs that have the same computational requirements and take unit time to execute. Each computer is modeled as an M/M/1 queueing system (i.e. the Poisson arrivals and exponentially distributed processing times) [42] and is characterized by its average processing rate μ_i, $i = 1, \ldots, n$. Jobs are generated by the users with a total average arrival rate Φ. The total job arrival rate Φ must be less than the aggregate processing rate of the system (i.e. $\Phi < \sum_{i=1}^{n} \mu_i$). The system model is presented in Figure 1. The computers must collaborate in making the decision on how to split the flow of jobs arriving at the system such that the job expected response time at each individual computer is minimized. The decision on how the flow of jobs is allocated should also guarantee a fair treatment of jobs. More specifically, jobs should have approximately equal expected response times at each computer independently of the computer to which they are allocated. The expected response time of jobs processed at computer i is given by

$$F_i(\beta_i) = \frac{1}{\mu_i - \beta_i}, \quad i = 1, \ldots, n \tag{4}$$

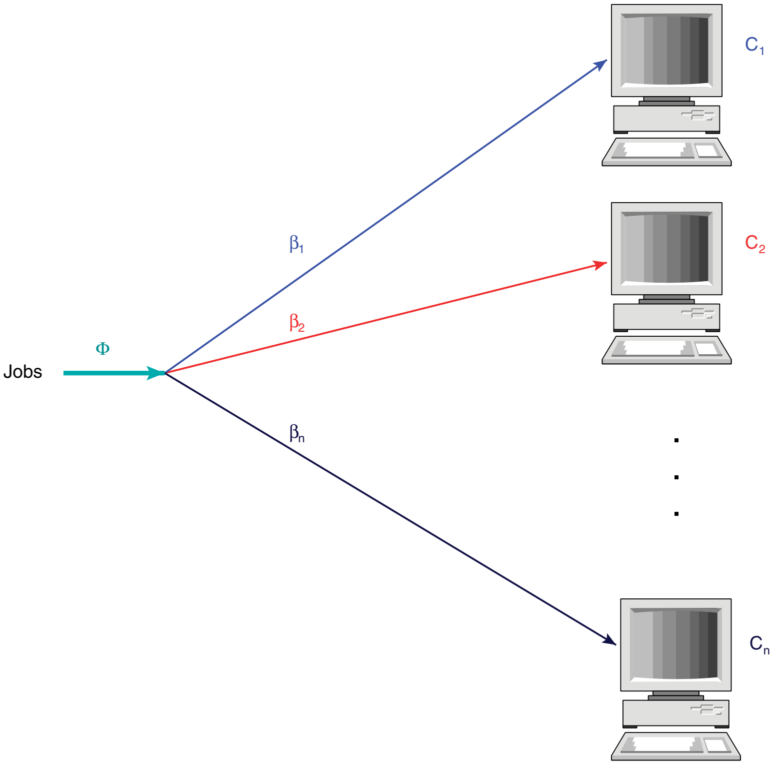

Figure 1. The distributed system model.

where β_i is the average arrival rate of jobs at computer i. Thus, the problem is to determine the fraction of the job flow (β_i) that each computer (i) must process. The cooperation among computers, which is needed to make the decision, is materialized by either peer-to-peer communication or by considering a central dispatcher that receives the processing rates from computers and makes the decision on their behalf according to an agreed algorithm.

We model the above load-balancing problem as a cooperative game in which each computer i is a player that minimizes the expected response time, i.e.

$$\min_{\beta_i} F_i(\beta_i), \quad i = 1, \ldots, n \tag{5}$$

In the description of the general game in the previous section we use a vector as the argument to the objective functions of the game. As F_i depends only on β_i we do not use the vector $\{\beta_i, \beta_2, \ldots, \beta_n\}$ as its argument.

The *load-balancing strategy* of computer i is represented by β_i. The minimization of the response time at each computer should be done such that the vector of load-balancing strategies $\{\beta_i, \beta_2, \ldots, \beta_n\}$ is feasible. A *feasible* vector of load-balancing strategies satisfies the following constraints:

$$\beta_i < \mu_i, \quad i = 1, \ldots, n \tag{6}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{7}$$

$$\beta_i \geq 0, \quad i = 1, \ldots, n \tag{8}$$

The first constraint is the 'stability' condition and the second one is the 'conservation law' for the system of M/M/1 queues used to model the set of computers. As minimizing $1/(\mu_i - \beta_i)$, $i = 1, \ldots, n$, is equivalent to maximizing $-\beta_i$, $i = 1, \ldots, n$, the optimization problem associated with the above game is

$$\max_{\beta_i}(-\beta_i), \quad i = 1, \ldots, n \tag{9}$$

subject to

$$-\beta_i > -\mu_i, \quad i = 1, \ldots, n \tag{10}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{11}$$

$$-\beta_i \leq 0, \quad i = 1, \ldots, n \tag{12}$$

We now formally define the cooperative load-balancing game among computers considering the model described above.

Definition 3.1 (*The cooperative load-balancing game*). The cooperative load-balancing game consists of

(i) n computers as *players*.
(ii) The *set of strategies*, X, is defined by the following constraints:

$$-\beta_i \geq -\mu_i, \quad i = 1, \ldots, n \tag{13}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{14}$$

$$-\beta_i \leq 0, \quad i = 1, \ldots, n \tag{15}$$

(iii) For each computer i, $i = 1, 2, \ldots, n$, the *objective function* $f_i(\beta_i) = -\beta_i$, $\beta_i \in X$. The goal is to maximize simultaneously all $f_i(\beta_i)$.
(iv) For each computer i, $i = 1, 2, \ldots, n$, the initial performance $u_i^0 = -\mu_i$. This is the value of f_i required by computer i without any cooperation to enter the game.

DOI: 10.1002/cpe

Remark. To satisfy the compactness requirement for X we allow the possibility that $\beta_i = \mu_i$. However, this requirement is eliminated because we assume that all n computers in the set J of players are able to achieve performance strictly superior to their initial performance. Let the initial agreement point be $u_i^0 = -\mu_i$, $i = 1, \ldots, n$. We assume that all computers agree that they will choose β_i such that $-\beta_i > -\mu_i$. This ensures that $\beta_i < \mu_i$. This is also the 'stability' condition for the M/M/1 system.

The cooperation among computers participating in this game is needed to obtain the initial agreement points. This can be accomplished by the computers themselves exchanging their processing rates or by involving a third party with which the computers communicate, in this case the job dispatcher.

In order to obtain a load-balancing scheme for the distributed system we need to solve the game defined above. The solution concept we adopt here is NBS. We next characterize NBS and then provide an algorithm that computes it. Our results are given in the following theorems and proposition.

Theorem 3.1. *For the load-balancing cooperative game defined above there is a unique bargaining point and the bargaining solutions are determined by solving the following optimization problem*:

$$\max_{\beta} \prod_{i=1}^{n} (\mu_i - \beta_i) \tag{16}$$

subject to

$$\beta_i < \mu_i, \quad i = 1, \ldots, n \tag{17}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{18}$$

$$\beta_i \geq 0, \quad i = 1, \ldots, n \tag{19}$$

Proof. Given in the appendix.

Theorem 3.2. *For the load-balancing cooperative game defined above the bargaining solution is determined by solving the following optimization problem*:

$$\max_{\beta} \sum_{i=1}^{n} \ln(\mu_i - \beta_i) \tag{20}$$

subject to

$$\beta_i < \mu_i, \quad i = 1, \ldots, n \tag{21}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{22}$$

$$\beta_i \geq 0, \quad i = 1, \ldots, n \tag{23}$$

Proof. Given in the appendix.

Remark. Theorem 3.2 implies that the solution (NBS) is Pareto optimal.

As a first step in obtaining the solution for our load-balancing cooperative game we solve the optimization problem given in Theorem 3.2 without requiring β_i $(i=1,\dots,n)$ to be non-negative. The solution of this problem is given in the following proposition.

Proposition 3.1. *The solution of the optimization problem in Theorem 3.2. without the constraint* $\beta_i \geq 0,\ i=1,\dots,n$ *is given by*

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n} \tag{24}$$

Proof. Given in the appendix.

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

Based on this fact and Proposition 3.1, we derive an algorithm (called COOP) for obtaining NBS for the load-balancing cooperative game. In the following we present this algorithm:

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

In order to execute COOP we need the values of the total average job arrival rate Φ and also the values of the processing rates μ_i, $i=1,\dots,n$. The value of the average job arrival rate at the system is estimated by considering the number of arrivals over a fixed interval of time (as presented in [43]). The value of the processing rates is estimated using the same technique by considering the number of jobs completed over a fixed interval of time.

COOP must be run periodically or when a change in load (Φ) is detected. We assume a single job dispatcher that runs on a dedicated computer. COOP can be implemented and executed by the dispatcher or it can be replicated and executed by every computer in the system.

In the first case, the computers send their processing rates ($\mu_i, i = 1, \ldots, n$) to the dispatcher that runs COOP. The processing rates determine the initial agreement point in the proposed cooperative load-balancing game. Once these are received the dispatcher executes COOP to compute the load allocation for each computer ($\beta_i,\ i = 1, \ldots, n$). Once the load allocation is computed the job dispatcher splits the incoming flow of jobs into n subflows, one for each computer. One commonly used technique for splitting the job flow is as follows [1]. When a new job arrives in the system it will be allocated to computer i with a probability given by β_i/Φ. Note that $\sum \beta_i/\Phi = 1$ from the stability condition given in (3).

In the second case, all the computers execute COOP concurrently. The computers exchange information about their processing rates (μ_i) only at the initialization of the system. Thus, before executing COOP an all-to-all broadcast communication step is required. This information exchange requires $O(n \log n)$ messages assuming that a hypercube-type all-to-all broadcast algorithm is used. This exchange of information is needed for the decision on the initial agreement points in the proposed cooperative load-balancing game. After this exchange each computer runs the COOP algorithm to compute its own allocation. Once the allocation is determined each computer requests its fraction of load from the dispatcher. Note that the dispatcher is needed only for sending the jobs from the flow of jobs it receives.

The following theorem proves the correctness of COOP algorithm.

Theorem 3.3 (Correctness). *The allocation* $\{\beta_1, \beta_2, \ldots, \beta_n\}$ *computed by the COOP algorithm solves the optimization problem in Theorem 3.2 and it is the NBS of the proposed cooperative load-balancing game.*

Proof. Given in the appendix.

Remark. The time complexity of this algorithm is $O(n \log n)$. This is mainly due to the procedure used to sort the computers which is $O(n \log n)$. Steps 2–4 take $O(n)$. If the algorithm is used to compute the allocation several times for the same distributed system then there is no need to perform the sorting operation each time. Thus, repeated use of the algorithm for the same distributed system requires only $O(n)$ operations for each use. In general, determining the NBS is an NP-hard problem [44]. Here it is possible to obtain an $O(n \log n)$ algorithm because in the proposed cooperative load-balancing game the player's objective functions are continuous and concave.

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

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

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

DOI: 10.1002/cpe

Remark. In some cases COOP does not allocate load to some of the slowest computers in the system. This is due to the fact that the fast computers' processing rates are so high that allocating more jobs to them and no jobs to slower computers yields a smaller expected response time for all jobs.

In order to characterize the fairness of load-balancing schemes we adopt here the *fairness index* proposed in [45]:

$$I(\mathbf{T}) = \frac{[\sum_{i=1}^{n} T_i]^2}{n \sum_{i=1}^{n} T_i^2} \tag{25}$$

Here $\mathbf{T} = (T_1, T_2, \ldots, T_n)$, where T_i is the expected response time of jobs that are processed at computer i. We selected this index because it satisfies the continuity property, which requires that any change in allocation should be reflected in the fairness metric. This property is not satisfied by other well-known fairness metrics such as min–max ratio [45].

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

For the proposed load-balancing cooperative game we can state the following.

Theorem 3.4 (Fairness). *The fairness index equals* 1 *when we use the COOP algorithm to solve the proposed load-balancing cooperative game.*

Proof. Given in the appendix.

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

Remarks. (i) Although COOP appears to be simple, it is very powerful since it provides a Pareto-optimal allocation of jobs to computers, it guarantees the fairness of allocation and it has a low computational complexity. (ii) COOP is a static algorithm. However, it can deal with dynamically changing loads if it is run periodically or when a change in load (Φ) is detected.

4. EXPERIMENTAL RESULTS

4.1. Simulation environment

The simulations were carried out using Sim + + [46], 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, pre-emption, and random number generation. The simulation model consists of a collection of computers connected by a communication network. Jobs arriving at the system are distributed by a central dispatcher to the computers according to the specified load-balancing scheme. Jobs that have been dispatched to a particular computer are *run-to-completion* (i.e. no pre-emption) in FCFS (first-come-first-served) order.

DOI: 10.1002/cpe

Each computer is modeled as an M/M/1 queueing system [42]. The main performance metrics used in our simulations are the *response time* and the *fairness index*. The simulations were run over several millions of seconds, sufficient to generate a total of one to two million 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.

We perform several types of experiments in order to study the impact of system utilization and heterogeneity on the performance of the proposed scheme. Because each of these experiments considers a different system configuration and simulation setup we describe the configuration and the setup in the subsections corresponding to each type of experiment.

4.2. Performance evaluation

For comparison purposes we implement two other representative static schemes: (i) a scheme that yields the system-wide optimal response time that is used as a baseline scheme for our experiments and (ii) a scheme that allocates jobs to computers in proportion to their computing power and yields the worst response time of the static schemes in the literature [47]. All the schemes considered in this study assume the existence of a job dispatcher that runs the load-balancing algorithm. A brief description of these schemes is given below:

Proportional scheme (*PROP*) [47]: According to this scheme jobs are allocated in proportion to the processing speed of computers. The computers estimate and report their average processing rates to the dispatcher. The dispatcher computes the load allocated to each computer and sends the jobs for execution. The following centralized algorithm is used for obtaining the load allocation.

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

This allocation seems to be a natural choice but it does not minimize the average response time of the system and it is not fair.

Overall optimal scheme (*OPTIM*) [1,2]: This scheme minimizes the expected response time over all jobs executed by the system. The loads at each computer (β_i) are obtained by solving the following nonlinear optimization problem:

$$\min \frac{1}{\Phi} \sum_{i=1}^{n} \beta_i F_i(\beta_i) \tag{26}$$

subject to the constraints

$$\beta_i < \mu_i, \quad i = 1, \ldots, n \tag{27}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{28}$$

$$\beta_i \geq 0, \quad i = 1, \ldots, n \tag{29}$$

DOI: 10.1002/cpe

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

OPTIM algorithm:

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

This scheme provides the *overall optimum* (global approach) for the expected response time. For this scheme, the expected response time of homogeneous jobs is the same for all jobs before they are allocated to computers but not after the allocation has been made. As the jobs do not obtain the same expected response time after they are allocated this scheme is not fair.

We next evaluate (via simulations) the schemes presented above under various system loads and configurations.

4.2.1. Effect of system utilization

To study the effect of system utilization we simulated a heterogeneous system consisting of 16 computers with four different processing rates. In Table I we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We choose 0.013 jobs/s as the processing rate for the slowest computer because it is a value that can be found in real distributed systems [1].

System utilization (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i} \tag{30}$$

Table I. System configuration.

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

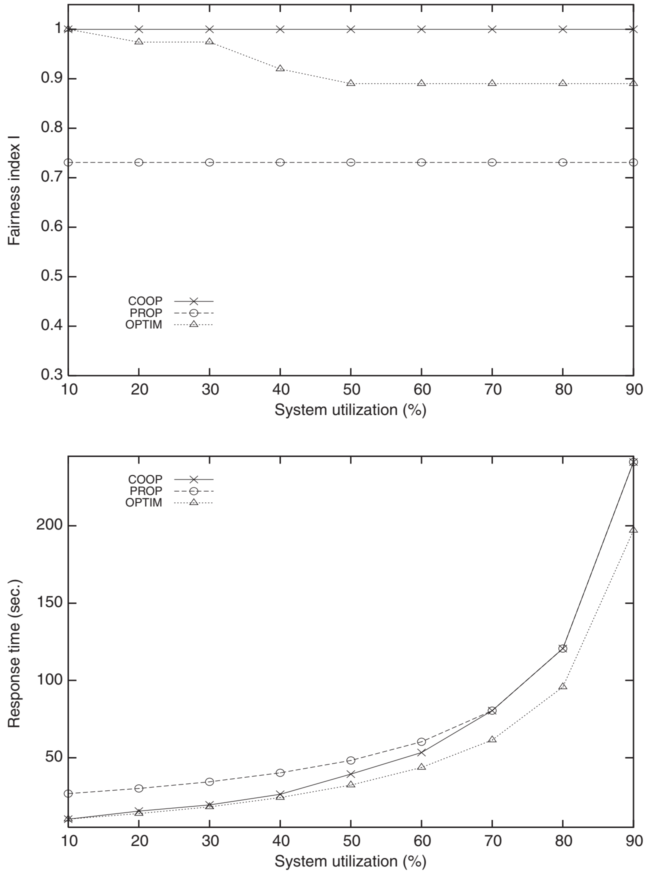

Figure 2. Fairness index and response time vs system utilization.

In Figure 2 we present the fairness index and the response time of the system for different values of system utilization ranging from 10 to 90%. This corresponds to a total arrival rate ranging from 0.066 to 0.596 jobs/s. In Table II we present the arrival rate and utilization of each computer at medium load ($\rho = 50\%$).

The main performance metric we consider here is the fairness index. The COOP scheme maintains a fairness index of 1 over the whole range of system loads and it is the only fair scheme here. It can be shown that PROP has a fairness index of 0.731, which is a constant independent of the system load. In the case of OPTIM, the expected response time for all jobs is the same before they are allocated to computers but not after the allocation has been made. This behavior of OPTIM is captured by the fairness index that varies from 1 at low load, to 0.88 at high load.

It can be observed that at low loads (ρ from 10 to 40%) all the schemes except PROP obtain almost the same response time. The poor response time of PROP scheme is due to the fact that the less powerful computers are significantly overloaded. At medium loads (ρ from 40 to 60%), the

Table II. Arrival rate and utilization of each computer at medium load ($\rho = 50\%$).

	COOP		PROP		OPTIM	
	Arrival rate	Utilization	Arrival rate	Utilization	Arrival rate	Utilization
1	0.1217	93.9	0.1040	79.8	0.1139	87.6
2	0.1217	93.2	0.1040	79.8	0.1139	87.4
3	0.0567	87.4	0.0520	79.9	0.0536	82.1
4	0.0567	87.4	0.0520	79.9	0.0536	82.0
5	0.0567	88.0	0.0520	80.1	0.0536	82.5
6	0.0177	67.4	0.0208	79.2	0.0188	72.2
7	0.0177	67.9	0.0208	80.8	0.0188	72.1
8	0.0177	67.1	0.0208	80.3	0.0188	73.3
9	0.0177	66.7	0.0208	81.1	0.0188	71.8
10	0.0177	68.2	0.0208	80.3	0.0188	72.7
11	0.0047	36.3	0.0104	79.7	0.0079	61.0
12	0.0047	37.8	0.0104	82.6	0.0079	61.2
13	0.0047	36.1	0.0104	79.9	0.0079	60.5
14	0.0047	36.7	0.0104	81.1	0.0079	59.9
15	0.0047	36.0	0.0104	80.7	0.0079	62.1
16	0.0047	35.1	0.0104	81.1	0.0079	61.8

COOP scheme provides significantly better response time than PROP being close to the response time of OPTIM. For example, at load level of 50% the response time of COOP is 19% less than PROP and 20% greater than OPTIM. At high loads, COOP and PROP yield the same response time, which is greater than that of OPTIM. COOP does not provide the global optimal response time as in the case of OPTIM but provides a Pareto-optimal allocation (see Remark following Theorem 3.2). At high loads, COOP allocates heavy loads to the fastest computers almost overloading them. It also allocates more load to the slowest computers compared with that allocated in the case of medium and low system loads. Overall, the effect is that at high loads the global response time obtained by COOP becomes the same as that of PROP. The fairness achieved by COOP comes at the cost of increasing the response time of the system. The increase in the response time of COOP necessary to achieve fairness varies from under 5% at low loads to under 25% at high loads.

An interesting issue related to the fairness is the impact of static load-balancing schemes on individual computers. In Figure 3 we present the response time at each computer for all static schemes at medium load ($\rho = 50\%$). Our scheme (COOP) guarantees equal response times for all computers. The value of the response time for each computer is equal to the value of overall response time (39.44 s). This means that jobs would have the same response time independent of the allocated computers. We can observe that some of the slowest computers are not utilized by the COOP scheme (C_{11}–C_{16}). The PROP scheme overloads the slowest computers and the overall response time is increased. The difference in the response time at C_1 (fastest) and C_{16} (slowest) is significant, 15 s compared with 155 s. In the case of OPTIM the response times are less balanced than COOP and unlike COOP the slowest computers are utilized. Still, the overall system response time is lower in OPTIM.

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

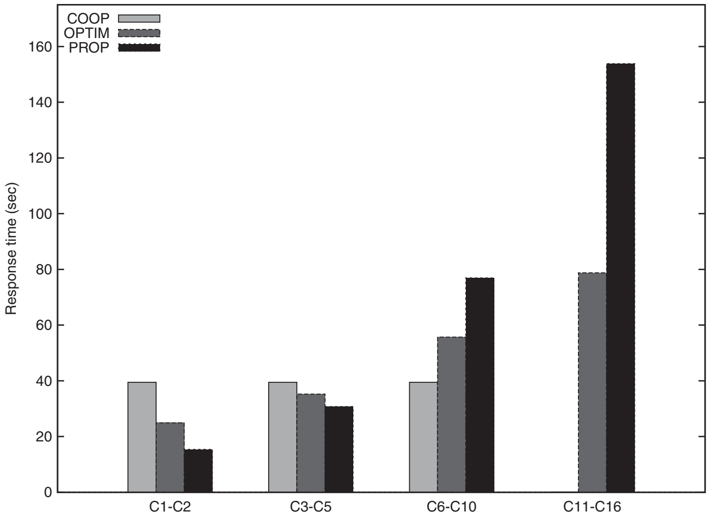

Figure 3. Response time at each computer (medium system load).

In Figure 4 we present the response time at each computer in the system when the system is heavily loaded ($\rho = 80\%$). The COOP scheme guarantees the same response time at each computer and utilizes all the computers. The difference in the response time between the fastest and slowest computers is large in the case of PROP (350 s) and moderate in the case of OPTIM (130 s). COOP scheme provides a fair and load balanced allocation, which is a desirable property in some systems.

4.2.2. Effect of heterogeneity

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

In this section we investigate the effectiveness of load-balancing schemes by varying the speed skewness. We simulate a system of 16 heterogeneous computers: 2 fast and 14 slow. The slow computers have a relative processing rate of 1 and we varied the relative processing rate of the fast computers from 1 (which correspond to a homogeneous system) to 20 (which correspond to a highly heterogeneous system). The system utilization was kept constant, $\rho = 60\%$. In Table III we present the processing rates of the computers in the system and the total arrival rates used in each type of experiment.

In Figure 5 we present the effect of speed skewness on the fairness and the response time. The COOP scheme maintains a fairness index of 1 over the whole range of system heterogeneity.

DOI: 10.1002/cpe

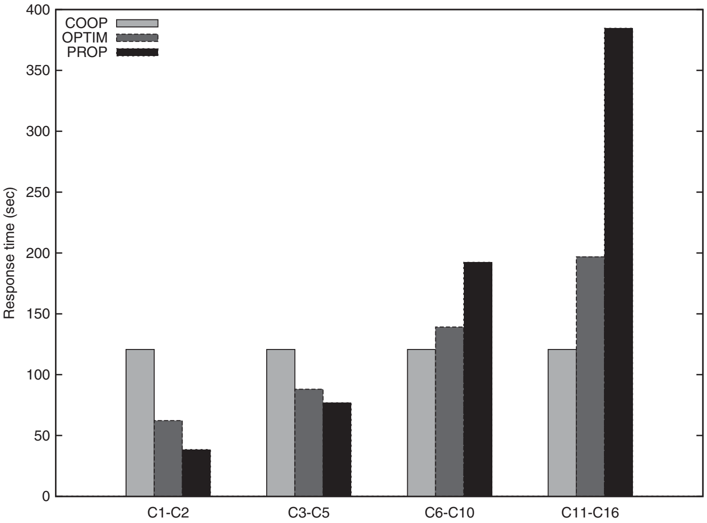

Figure 4. Response time at each computer (high system load).

Table III. System parameters.

Speed skewness	Processing rate of C_1, C_2 (jobs/s)	Processing rate of C_3–C_{16} (jobs/s)	Total arrival rate (jobs/s)
1	0.013	0.013	0.124
2	0.026	0.013	0.140
3	0.039	0.013	0.156
4	0.052	0.013	0.171
6	0.078	0.013	0.202
8	0.104	0.013	0.234
10	0.130	0.013	0.265
12	0.156	0.013	0.296
14	0.182	0.013	0.327
16	0.208	0.013	0.358
18	0.234	0.013	0.390
20	0.260	0.013	0.421

The fairness index of OPTIM varies from 0.98 for slightly heterogeneous systems to 0.92 for highly heterogeneous systems. The fairness of PROP is the worst among the static schemes considered here, reaching 0.88 for highly heterogeneous systems. This is due to the fact that COOP overloads the slow processors thus obtaining a higher response time at these processors.

It can be observed that increasing the speed skewness, OPTIM and COOP schemes yield low response times, which means that in highly heterogeneous systems COOP and OPTIM are very

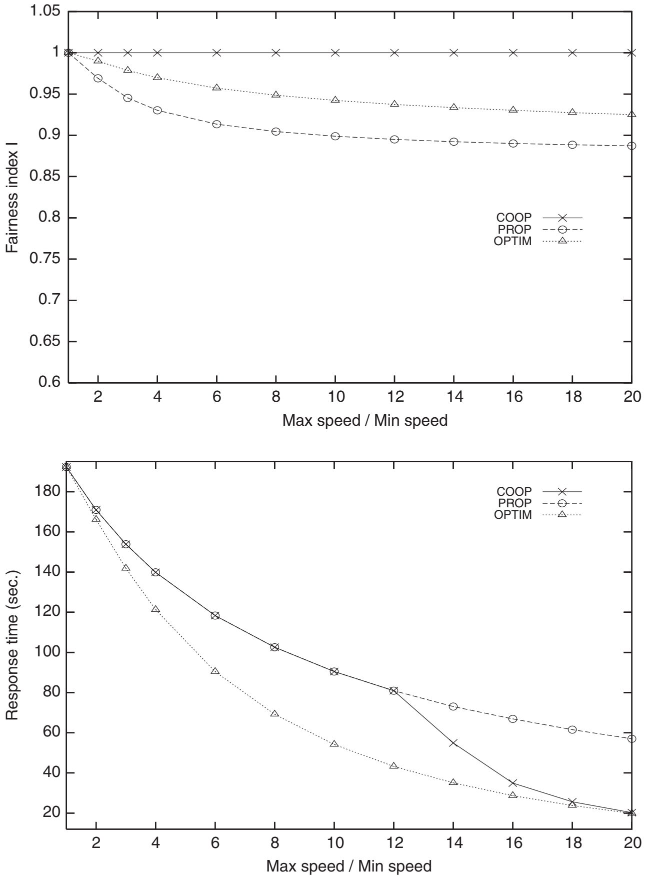

Figure 5. The effect of heterogeneity on the fairness index and the response time.

effective. COOP has the additional advantage of a fair allocation, which is very important in some systems. For systems with speed skewness less than 12, COOP and PROP yield the same response time. The improvement in the response time of COOP for speed skewness greater than 12 is due to the fact that COOP idles some of the slowest processors when the power of the fastest processors in the system is increased. PROP scheme performs poorly because it overloads the slowest computers.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a game-theoretic framework for obtaining a fair load-balancing scheme. The main goal was to derive a fair and Pareto-optimal allocation scheme. We formulated the load-balancing problem in single class job distributed systems as a cooperative game among computers. We showed that the *Nash Bargaining Solution* (NBS) of this game provides a Pareto-optimal operation point for the distributed system and it is also a fair solution. For the proposed cooperative load-balancing game we presented the structure of the NBS. Based on this structure we derived a new algorithm for computing it. We showed that the fairness index is always 1 using our new cooperative load-balancing scheme, which means that the solution is fair to all jobs. We compared by simulation the performance of our cooperative load-balancing scheme with that of other existing schemes. The simulations were run on a variety of system configurations that allowed us to compare the schemes in a fair manner and to obtain a high level of confidence on the results. As a general conclusion of the simulations we can state that COOP obtains an overall response time close to the one obtained by the global optimal scheme. The main advantage of COOP is that it treats all the jobs fairly in the sense that jobs would experience the same response time independently of the allocated computers. COOP is suitable for systems in which the fair treatment of the all users' jobs is an important consideration.

We plan to perform actual runs on distributed systems consisting of heterogeneous computers in order to validate our results. We plan to design and investigate load-balancing games that consider the cost of communication among computers in determining the load-balancing strategies. In addition, we will extend the techniques used in this paper to solve the load-balancing problem in multi-class job distributed systems where the jobs are grouped into different classes according to their sizes. Another interesting extension of this work will be to design a mechanism based on the proposed load-balancing scheme that takes into account the selfish behavior of the machines and the jobs. Future work will also 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.

APPENDIX A

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

Proof of Theorem 3.1. In Theorem 2.1 we consider $f_i(\beta_i) = -\beta_i$ that are concave and bounded above. The set defined by the constraints is convex and compact. Thus, the conditions in Theorem 2.1 are satisfied and the result follows. □

Proof of Theorem 3.2. Using the fact that $f_i(\beta_i) = -\beta_i$ are one-to-one functions of β_i and applying Theorem 2.2 the result follows. □

Proof of Proposition 3.1. The constraints in Theorem 3.2 are linear in β_i and $f_i(\beta_i) = -\beta_i$ has continuous first partial derivatives. This imply that the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality [48].

Let $\alpha \leq 0$, $\eta_i \leq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [48]. The Lagrangian is

$$L(\beta_i, \alpha, \eta_i) = \sum_{i=1}^{n} \ln(\mu_i - \beta_i) + \alpha \left(\sum_{i=1}^{n} \beta_i - \Phi \right) + \sum_{i=1}^{n} \eta_i (\beta_i - \mu_i) \tag{A1}$$

DOI: 10.1002/cpe

The first-order Kuhn–Tucker conditions are

$$\frac{\partial L}{\partial \beta_i} = -\frac{1}{\mu_i - \beta_i} + \alpha + \eta_i = 0, \quad i = 1, \ldots, n \tag{A2}$$

$$\frac{\partial L}{\partial \alpha} = \sum_{i=1}^{n} \beta_i - \Phi = 0 \tag{A3}$$

$$\mu_i - \beta_i \geq 0, \quad \eta_i(\mu_i - \beta_i) = 0, \;\; \eta_i \leq 0, \;\; i = 1, \ldots, n \tag{A4}$$

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

The solution of these conditions is

$$\frac{1}{\mu_i - \beta_i} - \alpha = 0, \quad i = 1, \ldots, n \tag{A5}$$

$$\sum_{i=1}^{n} \beta_i = \Phi \tag{A6}$$

By adding the equation given by (A5) for all i we obtain

$$n - \alpha \left(\sum_{j=1}^{n} \mu_j - \sum_{j=1}^{n} \beta_j \right) = 0 \tag{A7}$$

Solving for α we obtain

$$\alpha = \frac{n}{\sum_{j=1}^{n} \mu_j - \Phi} \tag{A8}$$

By replacing α in (A5), we obtain

$$\beta_i = \mu_i - \frac{\sum_{j=1}^{n} \mu_j - \Phi}{n} \tag{A9}$$

□

Lemma A.1. *Let $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_m$. If $\mu_m < (\sum_{j=1}^{m} \mu_j - \Phi)/m$ then the objective function from Theorem 3.2 is maximized, subject to the extra constraint $\beta_m \geq 0$, when $\beta_m = 0$.*

Proof. We add a new condition to the first-order conditions (A2)–(A4). The new set of conditions is as follows:

$$\frac{\partial L}{\partial \beta_i} = -\frac{1}{\mu_i - \beta_i} + \alpha + \eta_i = 0, \quad i = 1, \ldots, m - 1 \tag{A10}$$

$$\frac{\partial L}{\partial \beta_m} = -\frac{1}{\mu_m - \beta_m} + \alpha + \eta_m + \gamma_m = 0 \tag{A11}$$

DOI: 10.1002/cpe

$$\frac{\partial L}{\partial \alpha}=\sum_{i=1}^{m}\beta_i-\Phi=0 \tag{A12}$$

$$\mu_i-\beta_i\geq 0,\quad \eta_i(\mu_i-\beta_i)=0,\ \ \eta_i\leq 0,\ \ i=1,\ldots,m \tag{A13}$$

$$\beta_m\geq 0,\quad \gamma_m\beta_m=0,\quad \gamma_m\leq 0 \tag{A14}$$

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

$$\frac{1}{\mu_i-\beta_i}-\alpha=0,\quad i=1,\ldots,m-1 \tag{A15}$$

$$\frac{1}{\mu_m-\beta_m}-\alpha+\gamma_m=0 \tag{A16}$$

$$\sum_{i=1}^{m}\beta_i=\Phi \tag{A17}$$

$$\beta_m\geq 0,\quad \gamma_m\beta_m=0,\quad \gamma_m\leq 0 \tag{A18}$$

From Equation (A18) the value of β_i is either zero or positive. We consider each case separately.

Case 1: $\beta_m>0$. Equation (A18) implies that $\gamma_m=0$ and the solution to the conditions is

$$\beta_i=\mu_i-\frac{\sum_{j=1}^{m}\mu_j-\Phi}{m},\quad i=1,\ldots,m \tag{A19}$$

Case 2: $\beta_m=0$. It follows from Equation (A18) that $\gamma_m\leq 0$.

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

Proof of Theorem 3.3. The while loop in step 3 finds the minimum index m for which $\mu_m<(\sum_{j=1}^{m}\mu_j-\Phi)/m$.

If $m=n$ (that means that all β_i are positive) we apply Proposition 3.1 and the result follows.

If $m<n$ we have $\mu_i<(\sum_{j=1}^{i}\mu_j-\Phi)/i$ for $i=m,\ldots,n$. According to Lemma A.1 $\beta_m,\ldots,\beta_n$ must be 0 in order to maximize the objective function from Theorem 3.2. The algorithm sets $\beta_i=0$ for $i=m,\ldots,n$ in the while loop.

Because $\mu_m<(\sum_{j=1}^{m}\mu_j-\Phi)/m$ implies $0<\mu_m m\leq\sum_{j=1}^{m}\mu_j-\Phi$ and then $\Phi<\sum_{j=1}^{m}\mu_j$, we can apply Proposition 3.1 to the first m indices (that corresponds to the first m fast computers) and the values of the load allocation $\{\beta_1,\beta_2,\ldots,\beta_m\}$ maximizes the objective function and are given by

$$\beta_i=\mu_i-\frac{\sum_{j=1}^{m}\mu_j-\Phi}{m},\quad i=1,\ldots,m \tag{A20}$$

This is done in step 4. All β_i, for $i=1,\ldots,m$ are guaranteed to be non-negative because $\mu_i>(\sum_{j=1}^{m}\mu_j-\Phi)/m$. The load allocation $\{\beta_1,\beta_2,\ldots,\beta_m\}$ is the solution to the optimization problem in Theorem 3.2. According to Theorem 3.2 this is also the NBS of our cooperative load-balancing game. □

DOI: 10.1002/cpe

Proof of Theorem 3.4. Using Proposition 3.1, T_i for all n_a allocated computers ($\beta_i \neq 0$, $i = 1, \dots, n_a$) can be expressed as

$$T_i = \frac{1}{(\mu_i - \beta_i)} = \frac{1}{\mu_i - \mu_i + \frac{\sum_{j=1}^{n_a} \mu_j - \Phi}{n_a}} = \frac{n_a}{\sum_{j=1}^{n_a} \mu_j - \Phi} \tag{A21}$$

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

ACKNOWLEDGEMENTS

The authors wish to express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of this paper. This research was supported, in part, by research grants from (1) NASA NAG 2-1383, (2) NSF Grant ASC-9634775, (3) NSF Grant CCR-0312323 and (4) NSF Grant DGE-0654014.

REFERENCES

1. Tang X, Chanson ST. Optimizing static job scheduling in a network of heterogeneous computers. *Proceedings of the International Conference on Parallel Processing*, Toronto, Canada, August 2000; 373–382.
2. Tantawi AN, Towsley D. Optimal static load balancing in distributed computer systems. *Journal of the ACM* 1985; **32**(2):445–465.
3. Ross KW, Yao DD. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM* 1991; **38**(3):676–690.
4. Kameda H, Li J, Kim C, Zhang Y. *Optimal Load Balancing in Distributed Computer Systems*. Springer: London, 1997.
5. Fudenberg D, Tirole J. *Game Theory*. The MIT Press: 1994.
6. Dubey P. Inefficiency of Nash equilibria. *Mathematics of Operations Research* 1986; **11**(1):1–8.
7. Grosu D, Chronopoulos AT, Leung MY. Load balancing in distributed systems: An approach using cooperative games. *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, 2002; 52–61.
8. Kim C, Kameda H. An algorithm for optimal static load balancing in distributed computer systems. *IEEE Transactions on Computers* 1992; **41**(3):381–384.
9. Li J, Kameda H. A decomposition algorithm for optimal static load balancing in tree hierarchy network configuration. *IEEE Transactions on Parallel and Distributed Systems* 1994; **5**(5):540–548.
10. Li J, Kameda H. Optimal static load balancing in star network configurations with two-way traffic. *Journal of Parallel and Distributed Computing* 1994; **23**(3):364–375.
11. Kim C, Kameda H. Optimal static load balancing of multi-class jobs in a distributed computer system. *Proceedings of the 10th International Conference on Distributed Computing Systems*, Paris, France, May 1990; 562–569.
12. Lee H. Optimal static distribution of prioritized customers to heterogeneous parallel servers. *Computers & Operations Research* 1995; **22**(10):995–1003.
13. Li J, Kameda H. Load balancing problems for multiclass jobs in distributed/parallel computer systems. *IEEE Transactions on Computers* 1998; **47**(3):322–332.
14. Ni LM, Hwang K. Adaptive load balancing in a multiple processor system with many job classes. *IEEE Transactions on Software Engineering* 1985; **SE-11**(5):491–496.
15. Bharadwaj V, Ghose D, Mani V, Robertazzi TG. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press: Los Alamitos, CA, U.S.A., 1996.
16. Li X, Bharadwaj V, Ko C. Distributed image processing on a network of workstations. *International Journal of Computers and their Applications* 2003; **25**(2):1–10.
17. Blazewicz J, Drozdowski M, Markiewicz M. Divisible task scheduling—concept and verification. *Parallel Computing* 1999; **25**(1):87–98.
18. Chan S, Bharadwaj V, Ghose D. Large matrix–vector products on distributed bus networks with communication delays using the divisible load paradigm: Performance and simulation. *Mathematics and Computers in Simulation* 2001; **58**:71–92.

DOI: 10.1002/cpe

19. Bethel W, Tierney B, Lee J, Gunter D, Lau S. Using high-speed WANs and network data caches to enable remote and distributed visualization. *SC2000*: *High Performance Networking and Computing*, Dallas, TX, November 2000.
20. Bharadwaj V, Barlas G. Access time minimization for distributed multimedia applications. *Multimedia Tools and Applications* 2000; **12**(2–3):235–256.
21. Grosu D, Chronopoulos AT. Noncooperative load balancing in distributed systems. *Journal of Parallel and Distributed Computing* 2005; **65**(9):1022–1034.
22. Penmatsa S, Chronopoulos AT. Price-based user-optimal job allocation scheme for grid systems. *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium* (*IPDPS 2006*), *3rd High Performance Grid Computing Workshop*, Rhodes Island, Greece, April 2006.
23. Roughgarden T. Stackelberg scheduling strategies. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, Heraklion, Crete, Greece, July 2001; 104–113.
24. Altman E, Basar T, Jimenez T, Shimkin N. Routing in two parallel links: Game-theoretic distributed algorithms. *Journal of Parallel and Distributed Computing* 2001; **61**(9):1367–1381.
25. Koutsoupias E, Papadimitriou C. Worst-case equilibria. *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, Trier, Germany, 1999; 404–413.
26. Mavronicolas M, Spirakis P. The price of selfish routing. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, Heraklion, Crete, Greece, July 2001; 510–519.
27. Orda A, Rom R, Shimkin N. Competitive routing in multiuser communication networks. *IEEE/ACM Transactions on Networking* 1993; **1**(5):510–521.
28. Roughgarden T, Tardos E. How bad is selfish routing? *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, Redondo Beach, CA, November 2000; 93–102.
29. Archer A, Tardos E. Truthful mechanism for one-parameter agents. *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, Las Vegas, NV, October 2001; 482–491.
30. Feigenbaum J, Papadimitriou C, Shenker S. Sharing the cost of multicast transmissions. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 2000; 218–227.
31. Nisan N, Ronen A. Algorithmic mechanism design. *Games and Economic Behaviour* 2001; **35**(1–2):166–196.
32. Touati C, Altman E, Galtier J. Generalized Nash bargaining solution for bandwidth allocation. *Computer Networks* 2006; **50**:3242–3263.
33. Mazumdar R, Mason LG, Douligeris C. Fairness in network optimal flow control: Optimality of product forms. *IEEE Transactions on Communications* 1991; **39**(5):775–782.
34. Yaiche H, Mazumdar RR, Rosenberg C. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking* 2000; **8**(5):667–678.
35. Amazon Elastic Compute Cloud. http://www.amazon.com/ [1 March 2008].
36. On-demand Computing Using Network.com, White paper, Sun Microsystems, Inc., August 2007.
37. Mas-Collel A, Whinston MD, Green JR. *Microeconomic Theory*. Oxford University Press: New York, 1995.
38. Papadimitriou CH, Yannakakis M. On the approximability of trade-offs and optimal access of web sources. *Proceedings of the 41th IEEE Symposium on Foundations of Computer Science*, Redondo Beach, CA, November 2000; 86–92.
39. Nash J. The bargaining problem. *Econometrica* 1950; **18**(2):155–162.
40. Muthoo A. *Bargaining Theory with Applications*. Cambridge University Press: Cambridge, U.K., 1999.
41. Stefanescu A, Stefanescu MV. The arbitrated solution for multi-objective convex programming. *Revue Roumaine de Mathématique Pures et Appliquées* 1984; **29**:593–598.
42. Kleinrock L. *Queueing Systems—Volume 1*: *Theory*. Wiley: New York, 1975.
43. Anand L, Ghose D, Mani V. ELISA: An estimated load information scheduling algorithm for distributed computing systems. *Computers & Mathematics with Applications* 1999; **37**:57–85.
44. Deng X, Papadimitriou C. On the complexity of cooperative solution concepts. *Mathematics of Operations Research* 1994; **19**(2):257–266.
45. Jain RK, Chiu DM, Hawe WR. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. *Technical Report DEC-TR-301*, Digital Equipment Corporation, Eastern Research Lab, 1984.
46. Cubert RM, Fishwick P. *Sim + + Reference Manual*. CISE, University of Florida: Gainsville, FL, 1995.
47. Chow YC, Kohler WH. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Transactions on Computers* 1979; **C-28**(5):354–361.
48. Luenberger DG. *Linear and Nonlinear Programming*. Addison-Wesley: Reading, MA, 1984.

DOI: 10.1002/cpe