J. Parallel Distrib. Comput. 71 (2011) 537–555

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Game-theoretic static load balancing for distributed systems[☆]

Satish Penmatsa[a], Anthony T. Chronopoulos[b,*]

[a] Department of Math. & Computer Science, University of Maryland Eastern Shore, Princess Anne, MD 21853, United States

[b] Department of Computer Science, University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78249, United States

ARTICLE INFO

Article history:
Received 3 June 2009
Received in revised form
28 November 2010
Accepted 30 November 2010
Available online 17 December 2010

Keywords:
Load balancing
Distributed systems
Cooperative game
Non-cooperative game
Expected response time
Fairness

ABSTRACT

In this paper, we present a game theoretic approach to solve the static load balancing problem for single-class and multi-class (multi-user) jobs in a distributed system where the computers are connected by a communication network. The objective of our approach is to provide fairness to all the jobs (in a single-class system) and the users of the jobs (in a multi-user system). To provide fairness to all the jobs in the system, we use a cooperative game to model the load balancing problem. Our solution is based on the Nash Bargaining Solution (NBS) which provides a Pareto optimal solution for the distributed system and is also a fair solution. An algorithm for computing the NBS is derived for the proposed cooperative load balancing game. To provide fairness to all the users in the system, the load balancing problem is formulated as a non-cooperative game among the users who try to minimize the expected response time of their own jobs. We use the concept of Nash equilibrium as the solution of our non-cooperative game and derive a distributed algorithm for computing it. Our schemes are compared with other existing schemes using simulations with various system loads and configurations. We show that our schemes perform near the system optimal schemes and are superior to the other schemes in terms of fairness.

1. Introduction

A distributed system often consists of heterogeneous computing and communication resources. Due to the possible differences in the computing capacities and uneven job arrival patterns, the workload on different computers in the system can vary greatly [5,13]. Improving the performance of such a system by an appropriate distribution of the workload among the computers is commonly known as *load balancing*.

The load balancing schemes can be either *static* or *dynamic* [46]. The static schemes either do not use any system information or use only the average system behavior whereas the dynamic schemes consider instantaneous system states (runtime state information) in the job allocation calculations. However, as the overhead costs for the exchange of system state information increase, the static schemes can perform equally well or better compared to dynamic schemes [53]. The lower complexity or the minimal runtime overhead of the static schemes is also an added advantage. Another major drawback of the dynamic schemes is their sensitivity to inaccurate information used for job allocation purposes. Some dynamic allocations can result in extremely poor system performance even when the information accuracy is only slightly less than 100% [49,48]. Also, jobs in a distributed system can be divided into different classes based on their resource usage characteristics and ownership. Based on the number of job classes considered, we have a *single-class* or *multi-class* (*multi-user*) job distributed system.

In this paper, we consider the static load balancing problem for both single-class jobs and multi-user jobs in a distributed computer system that consists of heterogeneous host computers (nodes) interconnected by a communication network. Jobs arrive at each computer according to a time-invariant exponential process. Load balancing is achieved by transferring some jobs from nodes that are heavily loaded to those that are idle or lightly loaded. A communication delay will be incurred as a result of sending a job to a different computer for processing.

Since all the jobs belonging to the same user (or same class) usually have equal priority or are under the same administrative domain, we use a cooperative game to formulate the load balancing problem for single-class jobs. In a multi-class (or multi-user) job environment, jobs belong to various users and a user prefers to have her/his jobs executed first (or faster) than others. Because of this selfish nature, we use a non-cooperative game to model the load balancing problem for multi-user jobs. The *expected (mean) response time* of *a job* or *a user* or *the system* used in this paper is defined as the total time to execute a job or all the users jobs or

[☆] This work was supported in part by the National Science Foundation under grant number CCR-0312323. This research of A.T. Chronopoulos was partly supported by a NSF grant (HRD-0932339) to the University of Texas at San Antonio.

[*] Corresponding author.
E-mail addresses: spenmatsa@umes.edu (S. Penmatsa), atc@cs.utsa.edu (A.T. Chronopoulos).
URLs: http://www.umes.edu/mcs (S. Penmatsa), http://www.cs.utsa.edu/faculty/atc/ (A.T. Chronopoulos).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.11.016

all the jobs in the system which includes the processing time(s) at a node or nodes (processing delay), any queuing delays, and any communication delays [22].

1.1. Load balancing for single-class jobs

This load balancing problem is formulated as a cooperative game among the computers and the communication subsystem. The several decision makers (e.g., computers and the communication subsystem) cooperate in making decisions such that each of them will operate at its optimum. The decision makers have complete freedom of pre-play communication to make joint agreements about their operating points. Based on the *Nash Bargaining Solution* (NBS) which provides a Pareto optimal and fair solution, we provide an algorithm (CCOOP) for computing the NBS for our cooperative load balancing game. The objective of this cooperative load balancing scheme is to provide fairness to all the jobs, *i.e.* all the jobs (of approximately the same size) should experience approximately the same expected response time independent of the computers allocated for their execution.

1.2. Load balancing for multi-user jobs

This problem is formulated, taking into account the users' mean node delays and the mean communication delays, as a non-cooperative game among the users. Each user minimizes her/his own response time independently of the others and they all eventually reach an equilibrium. We use the concept of Nash equilibrium as the solution of our non-cooperative game and derive a distributed algorithm (NCOOPC) for computing it. The objective of this non-cooperative load balancing scheme is to provide fairness to all the users *i.e.* all the users should have approximately the same expected response time independent of the computers allocated for the execution of their jobs (of approximately the same size).

Remark 1.1. In the above we do not mean that the computers or the users engage in games, but, the load balancing problems will be solved using game theory models and the solution of the games will be used for job allocation.

1.3. Motivation and contribution

Most of the previous studies on static load balancing considered the minimization of the overall system expected response time as their main objective. However, some jobs or users may experience much longer expected response time than others in such allocations. Also, past load balancing algorithms whose objective is to provide fairness did not take the communication costs into account. In current distributed systems, especially Grid computing systems [10], the computing resources are distributed over the globe and so communication delays will be incurred because of job transfers which can play an important role in load balancing.

Here, we consider the static load balancing problem for both single-class jobs and multi-user jobs in a distributed computer system with the objective of providing fairness to all the jobs (in the single-class job system) and the users of the jobs (in the multi-user job system) by taking the communication costs into account. Fairness of allocation is an important factor in modern distributed systems and our schemes will be suitable for systems in which the fair treatment of the jobs or users is as important as other performance characteristics. Fairness is a major issue in many modern utility computing systems such as Amazon Elastic Compute Cloud [4] and Sun Grid Compute Utility [38] where users pay the 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.

To provide fairness to all the jobs in the system *i.e.* to find an allocation of jobs to computers that yields an equal or approximately equal expected response time for all the jobs (of approximately the same size), we use the framework provided by cooperative game theory. To provide fairness to all the users in the distributed system *i.e.* to find an allocation of users' jobs to computers that yields an equal or approximately equal expected response time for all the users (with jobs of approximately the same size), we use the framework provided by non-cooperative game theory.

We assume all jobs are of the same size in terms of the computation time required to be executed by the slowest computer. In the case where there exist jobs of unequal size then we assume that they are divisible. We thus assume that they are divided into jobs of the same size before they are scheduled for execution.

We perform simulations with various system loads and configurations to evaluate the performance of the proposed load balancing schemes. For comparison, we also implemented other representative static load balancing schemes. These static schemes are: (i) schemes that yield the system-wide optimal expected response time which are used as baseline schemes for our experiments (OPTIM [26] (which minimizes the expected response time of all the jobs in a single-class job system) and GOS [25] (which minimizes the expected response time of all the jobs in a multi-class job system)); and (b) schemes which allocate jobs to computers in proportion to their computing power and yield the worst expected response time of the static schemes in the literature (PROP [8] (which allocates the jobs to the computers in proportion to their processing speeds in a single-class job system) and PROP_M [8] (which allocates the users' jobs to the computers in proportion to their processing speeds in a multi-class job system)). We show that the proposed load balancing schemes not only provide fairness but also perform near the system-wide optimal load balancing schemes.

1.4. Related work

Extensive studies have been made on the static load balancing problem in single-class and multi-class job distributed systems ([51,26,31,50,22,30,32,33,37,43,1,6] and references there-in). Most of the above used the *global* approach, where the focus is on minimizing the expected response time of the entire system over all the jobs. Different network configurations were considered and the problem was formulated as a non-linear optimization problem and as a polymatroid optimization problem. The schemes that implement the global approach determine a load allocation to obtain a system-wide optimal response time and the fairness of allocation was not considered.

Load balancing for single-class jobs based on *cooperative* game theory has been studied in [20,19]. However, the communication costs were not taken into account in the above studies and the effect of *system size* and *communication time* on the proposed scheme were not studied. In this paper, we study game-theoretic load balancing schemes for both single-class and multi-class job systems by taking the communication costs into account and study the effect of system size and communication time on the proposed schemes.

Preliminary results based on cooperative game theory for single-class job systems by taking the communication subsystem into account can be found in [40]. Here, we evaluate the performance of the proposed cooperative scheme (CCOOP) using 32 computers compared to 16 computers in [40]. Also, in this paper, the effect of system size and the communication time on CCOOP are studied, the performance metrics and other implemented schemes are explained in more detail, and a numerical example illustrating CCOOP is provided. In this paper, we also study the NCOOPC load balancing scheme for multi-class job systems based on non-cooperative game theory. The objective of NCOOPC is to provide fairness to all the users in the system in terms of their *expected response time*. The performance of NCOOPC is evaluated with 32 computers and 20 users. The effect of system utilization, heterogeneity, system size, and communication time on NCOOPC are studied and a numerical example illustrating NCOOPC is also provided. The proof for computing the *best response* of the users for Nash equilibrium is also presented.

There exist only a few studies that use the *non-cooperative* approach for load balancing in distributed systems. Kameda et al. [22] derived load balancing algorithms for single and multi-class jobs using non-cooperative games based on Wardrop equilibrium. In this case each of the infinitely many jobs optimizes its own response time independently of the others and they all eventually reach an equilibrium. However, under certain conditions [11] this equilibrium load allocation provides a sub-optimal system-wide response time. Non-cooperative load balancing for finitely many jobs based on Nash equilibrium was studied in [18]. However, the communication subsystem was not taken into account. The problem was formulated as a Stackelberg game in [44] and was shown that it is NP-hard to compute the optimal Stackelberg allocation strategy. Extensive studies were made on the problem of routing traffic in networks using non-cooperative game models ([3,28,35,39,12,27] and references there-in) and game theory was also used to model grid systems ([29,23,45,49,48] and references there-in) and for price-based job allocation in distributed systems [16,17,54]. Preliminary results for job allocation in grid systems based on non-cooperative game theory by taking the communication subsystem into account can be found in [41].

In [15], a workload allocation policy (MMP) for heterogeneous systems is studied whose objective is to provide fairness to the jobs in the system. More specifically, MMP minimizes or eliminates the difference in expected response times at the fastest and slowest computers. It was shown that fairness is achieved at the expense of a tolerable increase in the overall system expected response time. MMP assumes that all the jobs are of the same type (or belong to the same class) and does not take into account the communication costs for transferring jobs.

CCOOP and NCOOPC (studied in this paper) provide fairness to the jobs and the users respectively and the performance of CCOOP and NCOOPC are very close to OPTIM and GOS respectively for low and medium system loads (OPTIM and GOS provide the minimum overall expected response time for single and multi-class job distributed systems respectively). CCOOP and NCOOPC perform significantly better than PROP and PROP_M respectively for high system loads (PROP and PROP_M are not optimal and in simulations they yield the worst overall expected response time for single and multi-class job distributed systems respectively).

In [49], a load balancing scheme (GT) based on game theory for computational grids was proposed. GT is dynamic in nature as it responds to changes in system states during runtime. Experimental results showed that GT provides fairness to the grid users but with an increase in the overall system expected response time. However, as the overheads for transferring the system state information between the nodes increase, the efficiency and fairness of GT decreases. In [49], GT was only compared with PROP_M (denoted by PS in [49]) and not with any optimal schemes (*e.g.* GOS).

Significant results on workload heterogeneity for task scheduling in distributed systems have been published ([24,7,2] and references there-in). In this paper, we consider a work model on job scheduling in heterogeneous computer systems. A job may consist of one or more tasks. A job is ready to be executed, when all its tasks are ready for execution. Jobs in this paper may belong to different users and differ in their arrival rates.

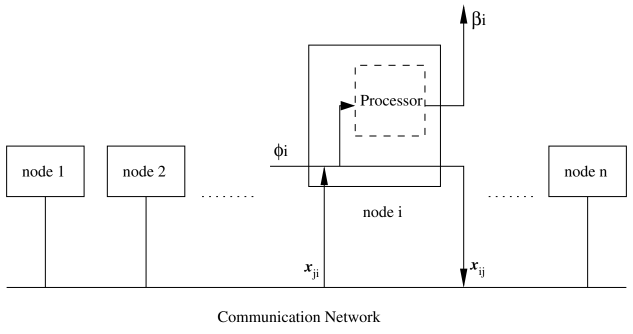


Fig. 1. Distributed system model for single-class jobs.

1.5. Organization

The rest of the paper is organized as follows. In Section 2, we study the load balancing for single-class jobs based on cooperative game theory. In Section 3, we study the load balancing for multi-user jobs based on non-cooperative game theory. The performance of the proposed cooperative and non-cooperative load balancing schemes is evaluated in Sections 4 and 5 respectively. Conclusions are drawn and future research directions are presented in Section 6.

2. Cooperative load balancing for single-class jobs

2.1. System model

We consider a distributed system model with n nodes (computers) connected by a communication network as shown in Fig. 1. The terminology and assumptions used are as follows: The nodes and the communication network are modeled as M/M/1 queuing systems [21]. In these queuing systems, the inter-arrival times and the service (processing) times are exponentially distributed and jobs arrive in a single queue (which is assumed to have infinite capacity) to a single computing resource with a First Come, First Served service discipline.

The following entities characterize M/M/1 queuing systems. We denote (i) the external job arrival rate at node i (*i.e.* the number of external jobs arriving at node i per unit time) by ϕ_i, (ii) the total external job arrival rate of the system (*i.e.* the total number of external jobs arriving into the system per unit time) by Φ (so, $\Phi = \sum_{i=1}^{n} \phi_i$), (iii) the maximum processing rate of node i (*i.e.* the maximum number of jobs that can be processed at node i per unit time) by μ_i, (iv) the job processing rate (or load) allocated by the load balancing scheme for node i (*i.e.* the number of jobs that are to be processed at node i per unit time) by β_i, and (v) the job flow rate from node i to node j (*i.e.* the number of jobs sent from i to j per unit time) by x_{ij}.

A job arriving at node i may be either processed at node i or transferred to another node j through the communication network for remote processing. The decision of transferring a job does not depend on the state of the system and hence is *static* in nature. A job transferred from node i to node j receives its service at node j and is not transferred to other nodes. If a node i sends (receives) jobs to (from) node j, node j does not send (receive) jobs to (from) node i.

The response time of a job in a system as above consists of a node delay (queuing delay + processing delay) at the processing node and also some possible communication delay incurred due to a job transfer. Let the mean node delay for a job at node i be denoted by $D_i(\beta_i)$. Modeling each node as an M/M/1 queuing system [21],

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

We assume that the expected communication delay from node i to node j is independent of the source-destination pair (i, j) but may depend on the total traffic through the network denoted by λ where $\lambda = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$. Let the mean communication delay for a job be denoted by $G(\lambda)$. Modeling the communication network as an M/M/1 queuing system [21],

$$G(\lambda) = \frac{t}{1 - t\lambda}, \quad \lambda < \frac{1}{t} \tag{2}$$

where t is the mean communication time for sending or receiving a job. Note that $D_i(\beta_i)$ and $G(\lambda)$ are increasing positive functions.

We also assume that the communication delay incurred as a result of sending a job directly from node i to node j is less than or equal to the sum of the delays from node i to node k and from node k to node j. Based on this, we classify the nodes in the following way similar to [51]:

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

The network traffic λ can be expressed in terms of the variable β_i as: $\lambda = \frac{1}{2} \sum_{i=1}^{n} |\phi_i - \beta_i|$.

We define the differential node delay (d_i), differential communication delay (g), and inverse of differential node delay (d_i^{-1}) as follows:

$$d_i(\beta_i) = \frac{\partial}{\partial \beta_i} \ln D_i(\beta_i) = \frac{1}{\mu_i - \beta_i} \tag{3}$$

$$g(\lambda) = \frac{\partial}{\partial \lambda} \ln G(\lambda) = \frac{t}{(1 - t\lambda)} \tag{4}$$

$$d_i^{-1}(x) = \begin{cases} \mu_i - \dfrac{1}{x}, & \text{if } x > \dfrac{1}{\mu_i} \\ 0, & \text{if } x \le \dfrac{1}{\mu_i}. \end{cases} \tag{5}$$

Remark 2.1. $d_i(\beta_i)$ and $g(\lambda)$ are increasing positive functions based on our assumptions on $D_i(\beta_i)$ and $G(\lambda)$.

2.2. Cooperative load balancing

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

$$f_i(X) = D_i(\beta_i) \tag{6}$$

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

$$f_{n+1}(X) = G(\lambda) \tag{7}$$

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

Definition 2.1 (*The Cooperative Load Balancing Game*). The cooperative load balancing game consists of:

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

$$\text{Stability}: \beta_i < \mu_i, \quad i = 1, \ldots, n \tag{8}$$

$$\text{Conservation}: \sum_{i=1}^{n} \beta_i = \sum_{i=1}^{n} \phi_i = \Phi, \tag{9}$$

$$\text{Positivity}: \beta_i \ge 0, \quad i = 1, \ldots, n. \tag{10}$$

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

Remark 2.2. In the above definition, we can assume that $\beta_i \le \hat{\mu}_i$ to satisfy the compactness requirement for X where $\hat{\mu}_i = \mu_i - \epsilon$ for a small $\epsilon > 0$. We ignore this condition for simplicity. We also assume that all the players in the above game are able to achieve performance strictly superior to their initial performance.

Theorem 2.1. *For the cooperative load balancing game defined above there is a unique bargaining point and the bargaining solutions are determined by solving the following optimization problem:*

$$\min_X \left[G(\lambda) \prod_{i=1}^{n} D_i(\beta_i) \right] \tag{11}$$

subject to the constraints (8)–(10).

Proof. In Appendix A. □

Theorem 2.2. *For the cooperative load balancing game defined above the bargaining solution is determined by solving the following optimization problem:*

$$\min_X \left[\sum_{i=1}^{n} \ln D_i(\beta_i) + \ln G(\lambda) \right] \tag{12}$$

subject to the constraints (8)–(10).

Proof. In Appendix A. □

Theorem 2.3. *The solution to the optimization problem in Theorem 2.2 satisfies the relations*

$$\begin{aligned} &d_i(\beta_i) \ge \alpha + g(\lambda), \quad \beta_i = 0 \ (i \in R_d), \\ &d_i(\beta_i) = \alpha + g(\lambda), \quad 0 < \beta_i < \phi_i \ (i \in R_a), \\ &\alpha + g(\lambda) \ge d_i(\beta_i) \ge \alpha, \quad \beta_i = \phi_i \ (i \in N), \\ &d_i(\beta_i) = \alpha, \quad \beta_i > \phi_i \ (i \in S), \end{aligned} \tag{13}$$

subject to the total flow constraint,

$$\sum_{i \in S} d_i^{-1}(\alpha) + \sum_{i \in N} \phi_i + \sum_{i \in R_a} d_i^{-1}(\alpha + g(\lambda)) = \Phi \tag{14}$$

where α is the Lagrange multiplier.

Proof. In Appendix A. □

The relations in Theorem 2.3 can be interpreted as follows: The differential node delays of all sinks are the same (*i.e.* α). The differential node delays of all active sources are equal; they consist of the differential node delay at a sink and the differential communication delay due to sending a job through the network to a sink. The differential node delay for neutrals is not less than the differential node delay of sinks but not greater than the differential node delay of active sources. The differential node delay for idle sources is not less than the differential node delay of active sources; this makes idle sources send all their jobs to sinks.

Since obtaining a closed form solution for α from Eq. (14) is not possible, we use a simple method such as a binary search to solve Eq. (14) iteratively for α as in [26]. This is described in the CCOOP algorithm below. Using the α in an iteration of the binary search, a set of sink ($S(\alpha)$) and source nodes ($R_d(\alpha)$ and $R_a(\alpha)$) are determined and it is checked whether the traffic from the set of source nodes ($\lambda_R(\alpha)$) equals the traffic into the set of sink nodes ($\lambda_S(\alpha)$), in which case an optimal α is found. The set of neutral nodes are denoted by $N(\alpha)$. ($S(\alpha)$, $R_d(\alpha)$, $R_a(\alpha)$, $N(\alpha)$, $\lambda_S(\alpha)$, and $\lambda_R(\alpha)$ are defined in Definition A.4 in Appendix A.) In the following, we present an algorithm (CCOOP) for obtaining the Nash Bargaining Solution for our cooperative load balancing game.

CCOOP algorithm:

```
Input:
    Node processing rates: $\mu_1, \mu_2, \dots \mu_n$;
    Node job arrival rates: $\phi_1, \phi_2, \dots \phi_n$;
    Mean communication time: $t$.
Output:
    Load allocation to the nodes: $\beta_1, \beta_2, \dots \beta_n$.
1. Initialization: $\beta_i \leftarrow \phi_i$; $i \in N$; $i = 1, \dots, n$.
2. Sort the computers such that $d_1(\phi_1) \leq d_2(\phi_2) \leq \dots \leq$
   $d_n(\phi_n)$. If $d_1(\phi_1) + g(0) \geq d_n(\phi_n)$, STOP. (No load balancing
   is required)
3. Determine $\alpha$ (using a binary search):
      $a \leftarrow d_1(\phi_1)$
      $b \leftarrow d_n(\phi_n)$
      while(1) do
         $\lambda_S(\alpha) \leftarrow 0$
         $\lambda_R(\alpha) \leftarrow 0$
         $\alpha \leftarrow \frac{a+b}{2}$
         Calculate: $S(\alpha), \lambda_S(\alpha), R_d(\alpha), R_a(\alpha)$, and $\lambda_R(\alpha)$
         (eqs. (72)–(76)) in the order given for $i = 1, \dots, n$
         If $(|\lambda_S(\alpha) - \lambda_R(\alpha)| < \epsilon)$ EXIT
         If $(\lambda_S(\alpha) > \lambda_R(\alpha))$
            $b \leftarrow \alpha$
            else
            $a \leftarrow \alpha$
4. Determine the loads on the computers:
      $\beta_i \leftarrow 0$,   for $i \in R_d(\alpha)$
      $\beta_i \leftarrow d_i^{-1}(\alpha + g(\lambda))$,   for $i \in R_a(\alpha)$
      $\beta_i \leftarrow d_i^{-1}(\alpha)$,   for $i \in S(\alpha)$
      $\beta_i \leftarrow \phi_i$,   for $i \in N(\alpha)$
```

The following remark describes the stopping criteria and the time complexity of CCOOP.

Remark 2.3. (i) In step 2, we STOP when the total (node + communication) time for a job to be transferred from a more powerful to a less powerful node exceeds the node time on the less powerful node, if the network traffic equals 0. This means that a job will run faster on the 'origin' node than if transferred to a different node.
(ii) The running time of this algorithm is $O(n \log n + n \log 1/\epsilon)$, where ϵ denotes the acceptable tolerance used for computing α in step 3 of the algorithm.

The following remark describes the implementation of CCOOP in practice.

Remark 2.4. The CCOOP algorithm must be run periodically or when the system parameters (system load) change in order to recompute a new load allocation. For example, the job arrival rate at a node can be estimated by considering the number of arrivals over a fixed interval of time. When the arrival rates change above some threshold, then the algorithm can be restarted to compute the new loads for each computer.

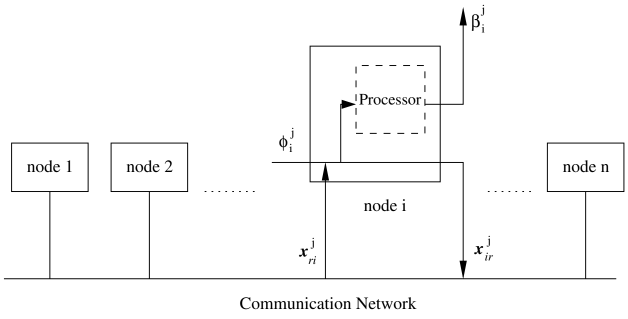


Fig. 2. Distributed system model for multi-user jobs.

The following example describes the CCOOP algorithm for a system of 3 nodes.

Example 2.1. In this example, we apply CCOOP algorithm to a system of 3 nodes. Let the processing rates of the nodes be $\mu_1 = 10$, $\mu_2 = 20$, and $\mu_3 = 40$. Let the job arrival rates to the nodes be $\phi_1 = 8$, $\phi_2 = 5$, and $\phi_3 = 2$. Let the mean communication time be 0.001 s. Step 1 initializes the loads on the nodes to $\beta_1 = 8$, $\beta_2 = 5$, and $\beta_3 = 2$. After sorting the nodes in step 2 we have $d_3(2) \leq d_2(5) \leq d_1(8)$ and $d_3(2)+g(0) = 0.026+0.001 = 0.027$. $d_1(8) = 0.5$. Since, $d_3(2)+g(0) < d_1(8)$ the algorithm proceeds to step 3. In step 3, α is determined using a binary search. Initial value of α will be 0.263 and the final value of α after exiting the 'while' loop is 0.04 and $\lambda_S = \lambda_R = \lambda = 13$ (ϵ is assumed to be 10^{-5}). Step 4 determines the final loads for the nodes as $\beta_1 = 0$, $\beta_2 = 0$, and $\beta_3 = 15$. Thus, node 3 is a sink and node's 1 and 2 are idle source nodes. □

3. Non-cooperative load balancing for multi-user jobs

3.1. System model

We consider a distributed system model as shown in Fig. 2. The system has n nodes (computers) connected by a communication network. The nodes and the communication network are modeled as M/M/1 queuing systems [21]. Jobs arriving at each node may belong to m different users.

The terminology and notations used are as follows: We denote (i) the external job arrival rate of user j to node i (*i.e.* the number of external jobs of user j arriving at node i per unit time) by ϕ_i^j, (ii) the total job arrival rate of user j by ϕ^j $\left(\text{so}, \phi^j = \sum_{i=1}^n \phi_i^j\right)$, (iii) the total job arrival rate of the system by Φ $\left(\text{so}, \Phi = \sum_{j=1}^m \phi^j\right)$, (iv) the maximum processing rate of node i (*i.e.* the maximum number of jobs that can be processed at node i per unit time) by μ_i, (v) the job processing rate (or load) of user j allocated by the load balancing scheme for node i (*i.e.* the number of jobs of user j that are to be processed at node i per unit time) by β_i^j, (vi) the vector of loads at node i from user's $1, \dots, m$ by $\beta_i = [\beta_i^1, \beta_i^2, \dots, \beta_i^m]^T$, (vii) the load vector of all nodes $i = 1, \dots, n$ (from all user's $1, \dots, m$) by $\beta = [\beta_1, \beta_2, \dots, \beta_n]^T$, (viii) the vector of loads of user k allocated to nodes $1, \dots, n$ by $\beta^k = [\beta_1^k, \beta_2^k, \dots, \beta_n^k]^T$, (ix) the job flow rate of user j from node r to node s (*i.e.* the number of jobs of user j sent from i to j per unit time) by x_{rs}^j, (x) the job traffic through the network of user j by λ^j $\left(\lambda^j = \sum_{r=1}^n \sum_{s=1}^n x_{rs}^j,\ \lambda = [\lambda^1, \lambda^2, \dots, \lambda^m]^T, \lambda = \sum_{j=1}^m \lambda^j\right)$, and (xi) the mean communication time for sending or receiving a job from one node to another for any user by t.

A job arriving at node i may be either processed at node i or transferred to a neighboring node j for remote processing through the communication network and is not transferred to any further nodes. The decision of transferring a job does not depend on the state of the system and hence is *static* in nature. If a node i sends (receives) jobs to (from) node j, node j does not send (receive) jobs to (from) node i.

For each user j, nodes are classified into the following as in [25]:

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

Assuming that each node is modeled as an M/M/1 queuing system, the mean node delay for an user j job processed at node i is given by:

$$F_i^j(\beta_i) = \frac{1}{\left(\mu_i - \sum_{k=1}^{m} \beta_i^k\right)}. \tag{15}$$

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

$$G^j(\lambda) = \frac{t}{\left(1 - t\sum_{k=1}^{m} \lambda^k\right)}, \quad \sum_{k=1}^{m} \lambda^k < \frac{1}{t}. \tag{16}$$

Remark 3.1. $F_i^j(\beta_i)$ and $G^j(\lambda)$ are increasing positive functions.

The network traffic of user j can be expressed in terms of the variable β_i^j as: $\lambda^j = \frac{1}{2}\sum_{i=1}^{n} |\phi_i^j - \beta_i^j|$.

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

$$\begin{aligned} D^j(\beta) &= \frac{1}{\phi^j}\sum_{i=1}^{n} \beta_i^j F_i^j(\beta_i) + \frac{\lambda^j}{\phi^j} G^j(\lambda) \\ &= \frac{1}{\phi^j}\sum_{i=1}^{n} \frac{\beta_i^j}{\left(\mu_i - \sum_{k=1}^{m} \beta_i^k\right)} + \frac{\lambda^j t}{\phi^j\left(1 - t\sum_{k=1}^{m} \lambda^k\right)}. \end{aligned} \tag{17}$$

3.2. Non-cooperative game among the users

In this section, we formulate the load balancing problem as a non-cooperative game among the users. We use the game theory terminology introduced in [18]. Each user j ($j = 1, \ldots, m$) must find the workload (β_i^j) that is assigned to computer i such that the expected response time of her/his own jobs ($D^j(\beta)$) is minimized. The vector $\beta^j = [\beta_1^j, \beta_2^j, \ldots, \beta_n^j]$ is called the load balancing strategy of user j ($j = 1, \ldots, m$) and the vector $\beta^* = [\beta^1, \beta^2, \ldots, \beta^m]$ is called the strategy profile of the load balancing game. The strategy of user j depends on the load balancing strategies of the other users.

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

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

A *Non-cooperative 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 her/his expected response time (D^j). Each user j prefers the strategy profile β^* to the strategy profile $\beta^{*\prime}$ if and only if $D^j(\beta^*) < D^j(\beta^{*\prime})$.

We need to solve the above game for our load balancing scheme. A solution can be obtained at the Nash equilibrium [14] which is defined as follows.

Definition 3.1 (*Nash Equilibrium*)**.** A *Nash equilibrium* of the load balancing game defined above is a strategy profile β^* such that for every user j ($j = 1, \ldots, m$):

$$\boldsymbol{\beta}^j \in \arg\min_{\tilde{\boldsymbol{\beta}}^j} D^j(\boldsymbol{\beta}^1, \ldots, \tilde{\boldsymbol{\beta}}^j, \ldots, \boldsymbol{\beta}^m). \tag{18}$$

At the Nash equilibrium, a user j cannot further decrease her/his expected response time by choosing a different load balancing strategy when the other users' strategies are fixed. The equilibrium strategy profile can be found when each user's load balancing strategy is a *best response* to the other users' strategies.

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

$$\min_{\boldsymbol{\beta}^j} D^j(\boldsymbol{\beta}) \tag{19}$$

subject to the constraints:

$$\beta_i^j \geq 0, \quad i = 1, \ldots, n \tag{20}$$

$$\sum_{i=1}^{n} \beta_i^j = \phi^j \tag{21}$$

$$\sum_{j=1}^{m} \beta_i^j < \mu_i, \quad i = 1, \ldots, n. \tag{22}$$

Remark 3.2. In finding the solution to BR^j, the strategies of all the other users are kept fixed and so the variables in BR^j are the workloads of user j, *i.e.* $\boldsymbol{\beta}^j = (\beta_1^j, \beta_2^j, \ldots, \beta_n^j)$.

In order to solve the optimization problem in Eq. (19), for each user j, we define the differential node delay (f_i^j), the differential communication delay (g^j), and the inverse of the differential node delay ($(f_i^j)^{-1}$) as follows:

$$f_i^j(\beta_i) = \frac{\partial}{\partial \beta_i^j}[\beta_i^j F_i^j(\beta_i)] = \frac{\mu_i^j}{(\mu_i^j - \beta_i^j)^2} \tag{23}$$

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

$$g^j(\lambda) = \frac{\partial}{\partial \lambda^j}[\lambda^j G^j(\lambda)] = \frac{t g_{-j}}{(g_{-j} - t\lambda^j)^2} \tag{24}$$

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

$$(f_i^j)^{-1}(\beta_i|_{\beta_i^j = x}) = \begin{cases} \left(\mu_i^j - \sqrt{\dfrac{\mu_i^j}{x}}\right), & \text{if } x > \dfrac{1}{\mu_i^j} \\ 0, & \text{if } x \leq \dfrac{1}{\mu_i^j}. \end{cases} \tag{25}$$

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

Theorem 3.1. *The solution to the optimization problem BR^j satisfies the relations*

$$\begin{aligned}
&f_i^j(\beta_i) \geq \alpha^j + g^j(\lambda), \qquad \beta_i^j = 0 \quad (i \in R_d^j),\\
&f_i^j(\beta_i) = \alpha^j + g^j(\lambda), \qquad 0 < \beta_i^j < \phi_i^j \quad (i \in R_a^j),\\
&\alpha^j + g^j(\lambda) \geq f_i^j(\beta_i) \geq \alpha^j, \qquad \beta_i^j = \phi_i^j \quad (i \in N^j),\\
&f_i^j(\beta_i) = \alpha^j, \qquad \beta_i^j > \phi_i^j \quad (i \in S^j),
\end{aligned} \tag{26}$$

subject to the total flow constraint,

$$\sum_{i\in S^j}(f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j}) + \sum_{i\in N^j}\phi_i^j + \sum_{i\in R_a^j}(f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j+g^j(\lambda)}) = \phi^j \tag{27}$$

where α^j is the Lagrange multiplier.

Proof. In Appendix B. □

The relations in Theorem 3.1 can be interpreted as follows: For each user j: the differential node delays of all sinks are the same; the differential node delays of all active sources are equal; they consist of the differential node delay at a sink and the differential communication delay due to sending a job through the network to a sink. The differential node delay for neutrals is not less than the differential node delay of sinks but not greater than the differential node delay of active sources; and the differential node delay for idle sources is not less than the differential node delay of active sources.

Since it is not possible to obtain a closed form solution for α^j from Eq. (27), we use a binary search to solve Eq. (27) iteratively for α^j similar to [25]. This is described in the BEST-RESPONSE algorithm below. Using the α^j in an iteration of the binary search, a set of sink ($S^j(\alpha^j)$) and source nodes ($R_d^j(\alpha^j)$ and $R_a^j(\alpha^j)$) for a user are determined and it is checked whether the traffic from the set of source nodes ($\lambda_R^j(\alpha^j)$) equals the traffic into the set of sink nodes ($\lambda_S^j(\alpha^j)$), in which case an optimal α^j is found. The set of neutral nodes are denoted by $N^j(\alpha^j)$. ($S^j(\alpha^j)$, $R_d^j(\alpha^j)$, $R_a^j(\alpha^j)$, $N^j(\alpha^j)$, $\lambda_S^j(\alpha^j)$, and $\lambda_R^j(\alpha^j)$ are defined in Definition B.1 in Appendix B.)

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

BEST-RESPONSE algorithm:

```
Input:   $\phi^j, \beta, \lambda, \mu_1, \ldots, \mu_n$.
Output:  $\beta^j$.
1. Initialization: $\beta_i^j \leftarrow \phi_i^j;\ i \in N^j;\ i = 1, \ldots, n$.
2. Sort the computers such that $f_1^j(\beta_1|_{\beta_1^j=\phi_1^j}) \leq \cdots \leq f_n^j(\beta_n|_{\beta_n^j=\phi_n^j})$. If $f_1^j(\beta_1|_{\beta_1^j=\phi_1^j}) + g^j(\lambda|_{\lambda^j=0}) \geq f_n^j(\beta_n|_{\beta_n^j=\phi_n^j})$,
   STOP. (No load balancing is required)
3. Determine $\alpha^j$ (using a binary search):
      $a \leftarrow f_1^j(\beta_1|_{\beta_1^j=\phi_1^j})$
      $b \leftarrow f_n^j(\beta_n|_{\beta_n^j=\phi_n^j})$
      while(1) do
         $\lambda_S^j(\alpha^j) \leftarrow 0$
         $\lambda_R^j(\alpha^j) \leftarrow 0$
         $\alpha^j \leftarrow \frac{a+b}{2}$
         Calculate: $S^j(\alpha^j)$, $\lambda_S^j(\alpha^j)$, $R_d^j(\alpha^j)$, $R_a^j(\alpha^j)$, and $\lambda_R^j(\alpha^j)$
         (eqs. (103)–(107)) in the order given for $i = 1, \ldots, n$
         If ($|\lambda_S^j(\alpha^j) - \lambda_R^j(\alpha^j)| < \epsilon$) EXIT
         If ($\lambda_S^j(\alpha^j) > \lambda_R^j(\alpha^j)$)
            $b \leftarrow \alpha^j$
         else
            $a \leftarrow \alpha^j$
4. Determine user j's loads on the computers:
      $\beta_i^j \leftarrow 0$,   for $i \in R_d^j(\alpha^j)$
      $\beta_i^j \leftarrow (f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j+g^j(\lambda)})$,   for $i \in R_a^j(\alpha^j)$
      $\beta_i^j \leftarrow (f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j})$,   for $i \in S^j(\alpha^j)$
      $\beta_i^j \leftarrow \phi_i^j$,   for $i \in N^j(\alpha^j)$
```

The following remark proves the correctness of BEST-RESPONSE algorithm.

Remark 3.3 (*Correctness of BEST-RESPONSE Algorithm*)**.** In the above BEST-RESPONSE algorithm, the 'while' loop in step 3 computes an optimal α^j which partitions the nodes into Idle Sources, Active Sources, Neutrals, and Sinks. Once the partition is known, the loads for user j on various nodes are computed in step 4. These are in accordance with Theorem 3.1. Thus the load balancing strategy computed by the BEST-RESPONSE algorithm solves the optimization problem BR^j and its solution is the best response strategy of user j.

The following remark describes the time complexity of BEST-RESPONSE algorithm.

Remark 3.4. The running time of BEST-RESPONSE algorithm is $O(n \log n + n \log 1/\epsilon)$, where ϵ denotes the tolerance used for computing α^j in step 3 of the algorithm. The available processing rate at each computer ("i") as seen by a user ("j") (*i.e.* μ_i^j in Eq. (23)) used in the above algorithm can be determined by statistical estimation of the run queue length of each node.

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

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

NCOOPC distributed load balancing algorithm:

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

1. Receive the current strategies of all the other users from the left neighbor.
2. If the message is a termination message, then pass the termination message to the right neighbor and EXIT.
3. Update the strategies (β^j) by calling the BEST-RESPONSE.
4. Calculate D^j (Eq. (17)) and update the error norm.
5. Send the updated strategies and the error norm to the right neighbor.

An important question is if such *best response*-based algorithms converge to the Nash equilibrium. There exists results about the convergence of such algorithms in the context of routing in parallel links [39,27]. For our load balancing game there exists a unique Nash equilibrium because the objective functions of the players are continuous, convex, and increasing. Orda et al. [39,27] proved that if the objective functions are continuous, convex, and increasing there exists a unique Nash equilibrium for the game. Our simulations of NCOOPC algorithm with different settings in Section 5 confirm the theoretical results.

The following remark describes the implementation of NCOOPC in practice.

Remark 3.5. In practice, the NCOOPC algorithm could be implemented by the scheduling agent (process) associated with each user. Users (or agents) will use the strategies that are computed at the Nash equilibrium and the system remains in equilibrium. This equilibrium is maintained until a new execution of the algorithm is initiated. The scheduling agent of a user communicates with the agents of other users and makes the allocation decisions. The NCOOPC algorithm can be restarted periodically by the scheduling agents when the system parameters (or system load) change above some threshold. When the job arrival rate at a node changes, then the job queue length at that node also changes. The scheduling agent can estimate the job arrival rate of the user to a node by considering the number of arrivals over a fixed interval of time.

The following example describes the NCOOPC and BEST-RESPONSE algorithms for a system of 3 nodes and 2 users.

Example 3.1. In this example, we apply the NCOOPC distributed load balancing algorithm to a system of 3 nodes and 2 users. Let the processing rates of the nodes be $\mu_1 = 5,\ \mu_2 = 10$, and $\mu_3 = 15$; the job arrival rates of user 1 to the nodes be $\phi_1^1 = 3,\ \phi_2^1 = 2$, and $\phi_3^1 = 7$; the job arrival rates of user 2 to the nodes be $\phi_1^2 = 1,\ \phi_2^2 = 5$, and $\phi_3^2 = 4$; and the mean communication time be 0.001 s.

In iteration 1 of NCOOPC, user 1 receives the initial strategies of user 2 (*i.e.* $\beta_1^2 = 1, \beta_2^2 = 5$, and $\beta_3^2 = 4$ (the initial strategies of each user are their own arrival rates)) and updates her/his strategies by calling the BEST-RESPONSE (BR) algorithm as follows: BR step 1 initializes user 1's loads on the computers to $\beta_1^1 = 3,\ \beta_2^1 = 2$, and $\beta_3^1 = 7$. After sorting the nodes in BR step 2 we have $f_2^1([2, 5]) < f_3^1([7, 4]) < f_1^1([3, 1])$ and $f_2^1([2, 5]) + g^1([0, 0]) = 0.55 + 0.001 < f_1^1([3, 1]) = 4.0$. So, the BR algorithm proceeds to step 3. In BR step 3, α^1 is determined using a binary search. Initial value of α^1 is 2.27 and the final value of α^1 after exiting the 'while' loop is 0.89 and $\lambda_S^1 = \lambda_R^1 = \lambda^1 = 1.11$ (ϵ is assumed to be 10^{-5}). BR step 4 determines the final loads for user 1 as $\beta_1^1 = 2.63,\ \beta_2^1 = 7.48$, and $\beta_3^1 = 1.88$. User 1 checks for the error norm and sends her/his updated strategies (loads) to user 2.

User 2 receives the current strategies of user 1 and updates her/his strategies by calling the BEST-RESPONSE (BR) algorithm as follows: BR step 1 initializes user 2's loads on the computers to $\beta_1^2 = 1,\ \beta_2^2 = 5$, and $\beta_3^2 = 4$. After sorting the nodes in BR step 2 we have $f_2^2([7.48, 5]) < f_3^2([1.88, 4]) < f_1^2([2.63, 1])$ and $f_2^2([7.48, 5]) + g^2([1.11, 0]) = 0.60 + 0.001 < f_1^2([2.63, 1]) = 1.31$. So, the BR algorithm proceeds to step 3. In BR step 3, α^2 is determined using a binary search. Initial value of α^2 is 0.96 and the final value of α^2 after exiting the 'while' loop is 0.81 and $\lambda_S^2 = \lambda_R^2 = \lambda^2 = 0.63$. BR step 4 determines the final loads for user 2 as $\beta_1^2 = 4.47, \beta_2^2 = 1.16$, and $\beta_3^2 = 4.36$. User 2 updates the error norm and sends her/his updated strategies (loads) and error norm to user 1.

In iteration 2 of NCOOPC, user 1 updates her/his loads by calling the BR algorithm using the loads of user 2 from iteration 1, checks for the error norm, and passes the updated loads to user 2. User 2 now updates her/his loads using the updated loads of user 1, updates the error norm, and passes the updated loads and error norm to user 1. This process continues until the desired error norm is reached (10 iterations in this example). The final loads of user 1 are $\beta_1^1 = 1.67,\ \beta_2^1 = 3.95$, and $\beta_3^1 = 6.37$, and the final loads of user 2 are $\beta_1^2 = 1.30,\ \beta_2^2 = 3.32$, and $\beta_3^2 = 5.37$. Thus, for user 1, node 1 is an active source, node 2 is a sink, and node 3 is an active source, and for user 2 node 1 is a sink, node 2 is an active source, and node 3 is a sink. □

4. Performance evaluation of CCOOP

We perform simulations to evaluate the performance of the CCOOP scheme. The system parameters that are used in the experiments below are obtained using Sim++ [9] simulation software package. The performance metrics used in our simulations are the *expected response time* and the *fairness index*. The *fairness index* [21],

$$I(\mathbf{D}) = \frac{\left[\sum_{i=1}^{n} D_i\right]^2}{n \sum_{i=1}^{n} D_i^2} \tag{28}$$

is used to quantify the fairness of load balancing schemes. Here the input $\mathbf{D}$ is the vector $\mathbf{D} = (D_1, D_2, \dots, D_n)$ where D_i is the expected response time of jobs that are processed at computer i. This index is a measure of the 'equality' of response times at different computers. If all the computers have the same expected job response time, then $I = 1$ and the system is 100% fair to all jobs and it is load-balanced. If the differences on D_i increase, I decreases and the load balancing scheme favors only some jobs.

We perform simulations to study the impact of system utilization, heterogeneity, system size, and communication time on the performance of the proposed scheme. We describe the system configuration and simulation setup for each of the above factors in the subsections corresponding to them. We also implemented the following static load balancing schemes for comparison purposes.

- Overall Optimal Scheme (OPTIM) [26]: This scheme minimizes the expected response time over all the jobs executed by the system. The loads (β_i) at each computer are obtained by solving the following non-linear optimization problem:

$$\min \frac{1}{\Phi}\left[\sum_{i=1}^{n} \beta_i D_i(\beta_i) + \lambda G(\lambda)\right], \tag{29}$$

subject to the constraints (8)–(10).

The centralized algorithm for obtaining the loads is given in [26]. This scheme provides a system optimal solution but is unfair.
- Proportional Scheme (PROP) [8]: This scheme allocates the jobs to the computers in proportion to their processing speeds as follows:

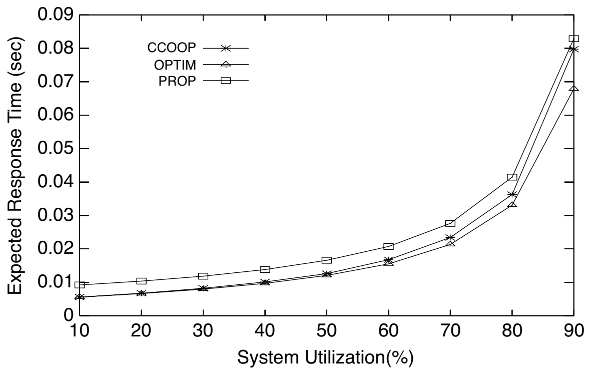
$$\beta_i \longleftarrow \Phi \frac{\mu_i}{\sum_{j=1}^{n} \mu_j}. \tag{30}$$

The allocation may not minimize the overall expected response time of the system and is unfair.

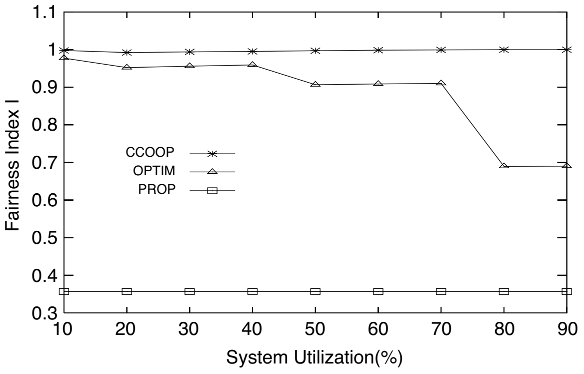
In the following we present and discuss the simulation results.

4.1. Effect of system utilization

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



(a) Expected response time.



(b) Fairness index.

Fig. 3. Effect of system utilization.

Table 1
System configuration.

Relative processing rate	1	5	10	15	20	25
Number of computers	6	6	6	4	4	6
Processing rate (jobs/s)	10	50	100	150	200	250

aggregate processing rate of the system:

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i}. \quad (31)$$

We simulated a heterogeneous system consisting of 32 computers to study the effect of system utilization. The system has computers with six different processing rates. The system configuration is shown in Table 1. The *relative processing rate* for a computer is defined as the ratio of its processing rate to the processing rate of the slowest computer in the system. 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. We chose fixed values for the system utilization and determined the total job arrival rate Φ. For example, if we consider $\rho = 10\%$ and an aggregate processing rate of 3860 jobs/s, then the total arrival rate in the system is $\Phi = 386$ jobs/s. The job arrival rate for each computer $\phi_i, i = 1, \dots, 32$ is determined from the total arrival rate as $\phi_i = q_i \Phi$, where the fractions q_i are given in Table 2. The mean communication time t is assumed to be 0.001 s.

Table 3 presents the job arrival rates to each computer (ϕ_i) and the job processing rates (departures rates or loads) at each computer (β_i) (using the notation ϕ_i/β_i) for system utilizations ranging from 10% to 90%. $\phi_i, i = 1, \dots, 32$ are calculated using the fractions given in Table 2 and Φ as described in the above paragraph. $\beta_i, i = 1, \dots, 32$ are obtained based on the CCOOP algorithm. The queue length for M/M/1 systems is infinite [21] and so we assumed that no jobs are lost $\left(\sum_{i=1}^{n} \phi_i = \sum_{i=1}^{n} \beta_i\right)$.

In Fig. 3(a), we present the expected 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 386 jobs/s to 3474 jobs/s (Table 3). It can be observed that CCOOP performs near the OPTIM for ρ ranging from 10% to 50% and is better than PROP for ρ ranging from 10% to 90%. For example, at 70% system utilization, the response time of CCOOP is around 16% less than that of PROP and around 9% greater than OPTIM. The poor performance of PROP is due to the fact that the less powerful computers are significantly overloaded. CCOOP does not provide a system optimal solution like OPTIM but provides a Pareto-optimal solution.

In Fig. 3(b), we present the fairness index for different values of system utilization. The CCOOP scheme has a fairness index of almost 1 for any system utilization. The fairness index of OPTIM drops from 0.98 at low load to 0.69 at high load and PROP maintains a fairness index of 0.35 over the whole range of system loads. The fairness achieved by CCOOP comes at the cost of increasing the response time of the system. This increased response time is still close to that of OPTIM as seen in Fig. 3(a) except for very high system loads.

Fig. 4 shows the expected response time at each computer for all the schemes at high system load ($\rho = 90\%$). CCOOP guarantees almost equal expected response times for all the computers. This means that the jobs will have the same expected response time

Table 2
Job arrival fractions q_i for each computer.

Computer	1–6	7–12	13	14–18	19–22	23–26	27–32
q_i	0.0025	0.01	0.02	0.025	0.04	0.05	0.07

Table 3
Job arrival rates/job processing rates (loads) for each computer (Ci) for various system utilizations.

System utilization (%)	C1–C6	C7–C12	C13	C14–C18	C19–C22	C23–C26	C27–C32
10	0.96/0	3.86/0	7.72/0	9.65/0	15.44/0	19.3/19.3	27.02/51.4
20	1.93/0	7.72/0	15.44/0	19.3/0	30.88/17.4	38.6/40.1	54.04/90.1
30	2.89/0	11.58/0	23.16/0	28.95/0	46.32/39.8	57.9/69.9	81.06/119.9
40	3.86/0	15.44/0	30.88/9.11	38.6/9.11	61.76/59.1	77.2/95.6	108.8/145.6
50	4.82/0	19.3/0	38.6/25.7	48.25/25.7	77.2/75.7	96.5/117.2	135.1/167.2
60	5.79/0	23.16/0	46.3/43.2	57.9/43.2	92.64/92.64	115.8/138.5	162.1/188.5
70	6.75/0	27.02/8.8	54.04/56.5	67.5/58.8	108.8/108.8	135.1/156.5	189.1/206.5
80	7.72/0	30.88/22.9	61.76/72.1	77.2/72.9	123.5/122.9	154.4/172.1	216.1/222.1
90	8.68/0	34.74/37.3	69.48/87.3	86.8/87.3	138.9/137.5	173.7/187.3	243.1/237.5

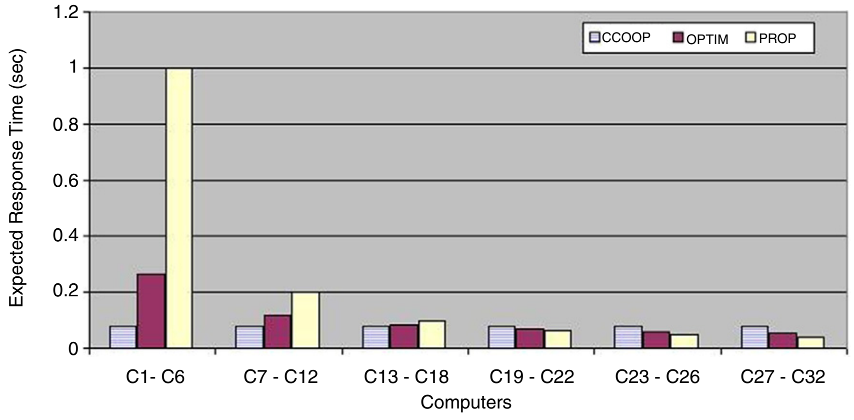


Fig. 4. Expected response time at each computer ($\rho = 90\%$).

Table 4
System parameters.

Speed skewness	1	4	16	36	64	100
μ_i of C1–C8	10	100	200	300	400	500
μ_i of C9–C24	10	50	50	50	50	50
μ_i of C25–C32	10	25	12.5	8.33	6.25	5
Φ (jobs/s)	192	1080	1500	1960	2430	2904

Table 5
Job arrival rates/job processing rates (loads) for each computer (Ci) for various skewness levels.

Speed skewness	C1–C8	C9–C24	C25–C32
4	38.33/76.9	38.33/27.6	20/2.68
16	97.5/160.5	40/13.4	10/0
36	157/242.4	40/1.28	8/0
64	217.7/303.7	40/0	6/0
100	279/362.9	40/0	4/0

independent of the allocated computers. In the case of OPTIM the expected response times are less balanced than CCOOP but the overall expected response time is lower than CCOOP as can be seen from Fig. 3(a). PROP overloads the slowest computers and the overall expected response time is increased. The difference in the expected response time at Computer's 1 through 6 (slowest) and Computer's 27 through 32 (fastest) is significant.

In the case of OPTIM and PROP, jobs are treated unfairly in the sense that a job allocated to the fast computer will have a low expected response time and a job allocated to a slow computer will have a high expected response time. CCOOP provides a fair and load balanced allocation which is desirable in many current distributed systems.

4.2. Effect of heterogeneity

In this section, we study the effect of heterogeneity on the performance of load balancing schemes. A simple way to characterize system heterogeneity is to use the processor speed. One of the common measures of heterogeneity is the *speed skewness* [50] which is defined as the ratio of maximum processing rate to minimum processing rate of the computers in the system. We study the effectiveness of load balancing schemes by varying the speed skewness.

We simulated a heterogeneous system of 32 computers (8 fast, 16 medium-fast, and 8 slow) to study the effect of heterogeneity. The slow computers have a relative processing rate of 1 and the relative processing rate of the fast and medium-fast computers is varied from 1 (homogeneous system) to 100 and 10 respectively (highly heterogeneous system). The system utilization was kept constant ($\rho = 60\%$) and the mean communication time t is assumed to be 0.001 s. In Table 4, we present the processing rates (μ_i jobs/s) of the computers in the different heterogeneous systems and the total arrival rates (Φ) of the systems. C1 through C8 represent the fast computers, C9 through C24 represent the medium-fast computers, and C25 through C32 represent the slow computers. The total arrival rates (Φ) are calculated using Eq. (31). Table 5 presents the job arrival rates to each computer (ϕ_i) and the job processing rates (departures rates or loads) at each computer (β_i) (using the notation ϕ_i/β_i) for various skewness levels. $\phi_i, i = 1, \ldots, 32$ are calculated as $\phi_i = q_i\Phi$, where q_i are fractions similar to that in Table 2. $\beta_i, i = 1, \ldots, 32$ are obtained based on the CCOOP algorithm.

Table 6
Total arrival rates.

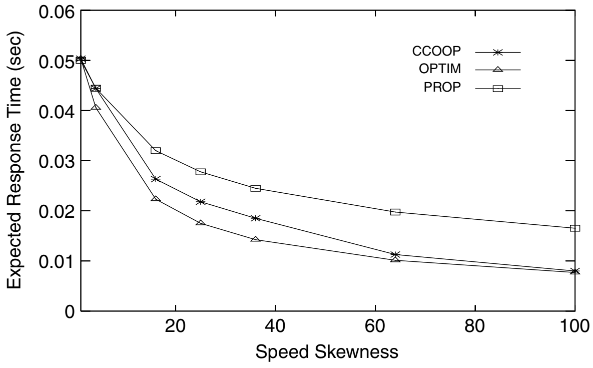
No. of computers	2	5	8	14	20	26	32
Φ (jobs/s)	120	168	216	312	408	504	600

Fig. 5(a) shows the effect of speed skewness on the expected response time. It can be observed that as the skewness increases, the performance of CCOOP approaches to that of OPTIM which means that in highly heterogeneous systems CCOOP is very effective. PROP performs poorly because it overloads the slow computers. The performance of CCOOP is increased with an increase in speed skewness because CCOOP idles some of the slowest computers when the power of the fastest computers in the system is increased.

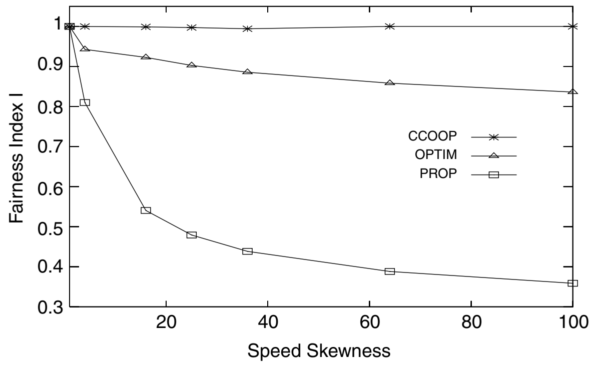
Fig. 5(b) shows the effect of speed skewness on the fairness index. It can be observed that CCOOP has a fairness index of almost 1 over all range of speed skewness. This shows that CCOOP guarantees almost equal expected response times for all the computers. The fairness index of OPTIM and PROP drops below 1 at low skewness to 0.83 and 0.35 respectively at high skewness.

4.3. Effect of system size

An important issue is to study the influence of system size on the performance of load balancing schemes. To study this issue we simulated a heterogeneous distributed system consisting of four types of computers: slow computers (relative processing rate = 1, 2 and 5) and fast computers (relative processing rate = 10). For a system size of 2, we used two fast computers. To increase the system size, we kept constant the number of fast computers and increased the number of slow computers. In Table 6, we present the total arrival rates for some of the experiments. The system

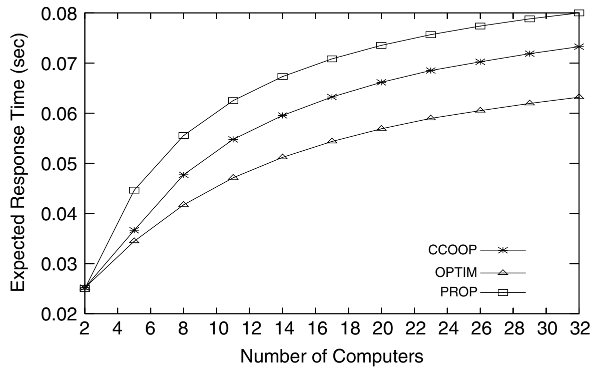


(a) Expected response time.

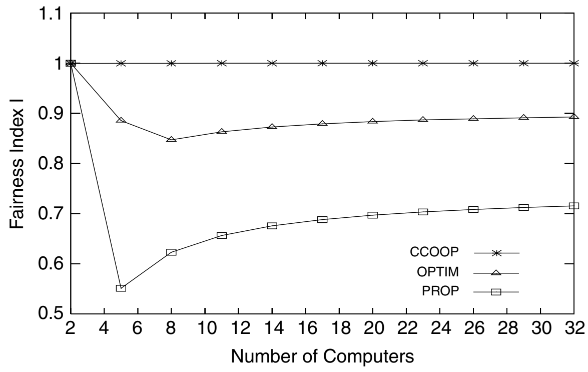


(b) Fairness index.

Fig. 5. Effect of heterogeneity.



(a) Expected response time.



(b) Fairness index.

Fig. 6. Effect of system size.

Table 7
Job arrival rates/job processing rates (loads) for computers for various system sizes.

System size	No. of computers × ϕ_i/β_i
2	$1 \times 80/59.1,\ 1 \times 40/60.8$
5	$2\times 52/72.4,\ 1\times 40/23.1,\ 1\times 15/0,\ 1\times 9/0$
8	$2\times 44/78.7,\ 2\times 40/29.2,\ 2\times 15/0,\ 2\times 9/0$
14	$2\times 28/82.9,\ 4\times 40/33.2,\ 4\times 15/3.2,\ 4\times 9/0$
20	$2\times 12/84.6,\ 6\times 40/34.8,\ 6\times 15/4.8,\ 6\times 9/0$
26	$2\times 36/85.6,\ 8\times 30/35.6,\ 8\times 15/5.9,\ 8\times 9/0$
32	$2 \times 30/86.2,\ 10 \times 30/36.2,\ 10 \times 15/6.5,\ 10 \times 9/0$

utilization was kept constant ($\rho = 60\%$) and the mean communication time t is assumed to be 0.001 s. The total arrival rates (Φ) are calculated using Eq. (31).

Table 7 presents ϕ_i and β_i (using the notation ϕ_i/β_i) (the computers are ordered in decreasing order of their speeds (actual processing rates)) for various system sizes presented in Table 6. $\phi_i, i = 1, \ldots, 32$ are calculated as $\phi_i = q_i\Phi$, where q_i are fractions similar to that in Table 2 and $\beta_i, i = 1, \ldots, 32$ are obtained based on the CCOOP algorithm.

Fig. 6(a) shows the expected response time where the number of computers increases from 2 to 32. It can be observed that the performance of CCOOP lies in between OPTIM and PROP. The sub-linear increase in the expected response time of CCOOP also shows that it is scalable.

Fig. 6(b) shows the fairness index for all the schemes as the system size increases. It can be observed that the CCOOP maintains a fairness index approximately equal to 1 with an increase in the system size. This means that the expected response times for all the computers is almost equal in the case of CCOOP. Thus, CCOOP provides an allocation which is fair to all the jobs independent of the allocated computers.

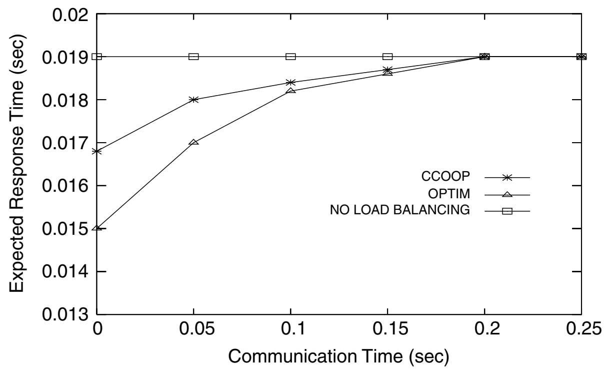
4.4. Effect of communication time

From the above, it can be observed that CCOOP scheme is not only fair, but also performs near the OPTIM for a low value of mean communication time t (low value of t represents a faster communications network). The effect of increasing t on the expected response time at medium system utilization ($\rho = 60\%$) for the configuration given in Table 1 is shown in Fig. 7(a).

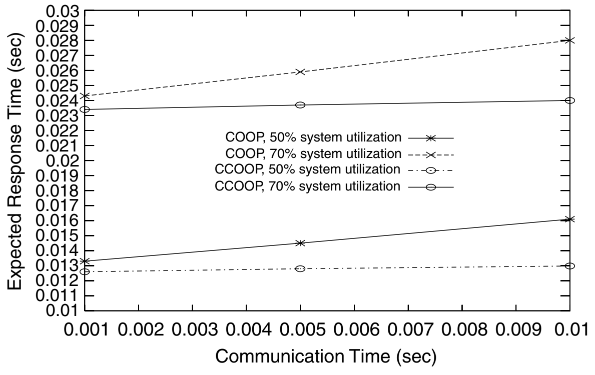
From Fig. 7(a), it can be observed that without load balancing (each computer processes its own local stream of jobs) the expected response time is 0.019 s. As t increases, the expected response time of CCOOP increases and reaches 0.019 s at $t = 0.2$ s, which is the limiting case of load balancing when the communication time is high ($t \geq 0.2$). This is because, as t increases, the load transfer from slow computers to fast computers decreases and the CCOOP scheme will not be able to load balance to its optimum. Similarly, the Fairness Index of CCOOP decreases as the communication time increases.

We also implemented the COOP scheme [19] for comparison. COOP is a cooperative game based load balancing scheme. However, it does not take the communication costs into account in finding the optimal solution that provides fairness.

Fig. 7(b) shows the effect of communication time on the performance of COOP and CCOOP at 50% and 70% system utilizations for the configuration given in Table 1. It can be observed that as t increases (from 0.001 to 0.01 s), CCOOP outperforms COOP. As t increases, the performance degradation of COOP is higher compared to that of CCOOP. For example, as



(a) CCOOP.



(b) COOP vs. CCOOP.

Fig. 7. Effect of communication time.

t increases from 0.001 to 0.01 s, the performance degradation of COOP is around 20% whereas the performance of CCOOP degrades by only around 3%. Although the figure shows plots for only a few values, the performance degradation of COOP compared to CCOOP is higher for any value of $t > 0$. This is because CCOOP takes the communication delays into account whereas COOP does not. Thus, CCOOP can show a significant performance improvement over COOP.

5. Performance evaluation of NCOOPC

We perform simulations to study the impact of system utilization, heterogeneity, system size, and communication time on the performance of NCOOPC scheme. The system parameters that are used in the experiments below are obtained using Sim++ [9] simulation software package. The performance metrics used are the *expected response time* and the *fairness index* [21]. The *fairness index* (defined from the users' perspective) is defined as,

$$I(\mathbf{C}) = \frac{\left[\sum_{j=1}^{m} C^j\right]^2}{m \sum_{j=1}^{m} C^{j^2}}. \tag{32}$$

The input $\mathbf{C}$ is the vector $\mathbf{C} = (C^1, C^2, \dots, C^m)$ where C^j is the expected response time of user j's jobs. If all the users have the same total expected response time, then $I = 1$ and the system is 100% fair to all users and it is load-balanced. If I decreases, then the load balancing scheme favors only some users. We also implemented the following load balancing schemes for comparison purposes:

- Global Optimal Scheme (GOS) [25]: This scheme minimizes the expected response time over all the jobs executed by the system to provide a system optimal solution. The loads (β_i^j) for each user are obtained by solving the following non-linear optimization problem:

$$\min D(\beta) = \frac{1}{\Phi} \sum_{j=1}^{m} \left[\sum_{i=1}^{n} \beta_i^j F_i^j(\beta_i) + \lambda^j G^j(\lambda) \right] \tag{33}$$

subject to the constraints (20)–(22).
- Proportional Scheme (PROP_M) [8]: According to this scheme each user allocates her/his jobs to computers in proportion to their processing rate as follows:

$$\beta_i^j \longleftarrow \phi^j \frac{\mu_i}{\sum_{k=1}^{n} \mu_k}. \tag{34}$$

Table 8
Job arrival fractions (q^j) for each user.

User	1–5	6–10	11–15	16–20
q^j	0.1	0.05	0.04	0.01

This allocation may not minimize the users' expected response times or the overall expected response time. The fairness index for this scheme is always 1 as can be easily checked from Eq. (32).

In the following we present and discuss the simulation results.

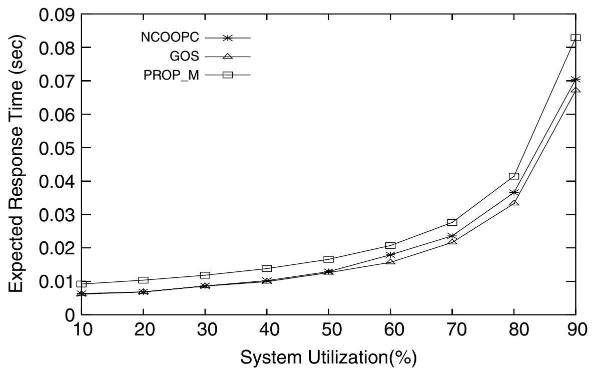
5.1. Effect of system utilization (ρ)

We simulated a heterogeneous system consisting of 32 computers to study the effect of system utilization. The system has computers with six different processing rates (Table 1) and is shared by 20 users. 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. The total job arrival rate Φ is chosen by fixing the system utilization. The job arrival rate for each user $\phi^j, j = 1, \dots, 20$ is determined from the total arrival rate as $\phi^j = q^j\Phi$, where the fractions q^j are given in Table 8. The job arrival rates of each user $j, j = 1, \dots, 20$ to each computer $i, i = 1, \dots, 32$, *i.e.* the ϕ_i^j's are obtained similar to Table 3. t is assumed to be 0.001 s.

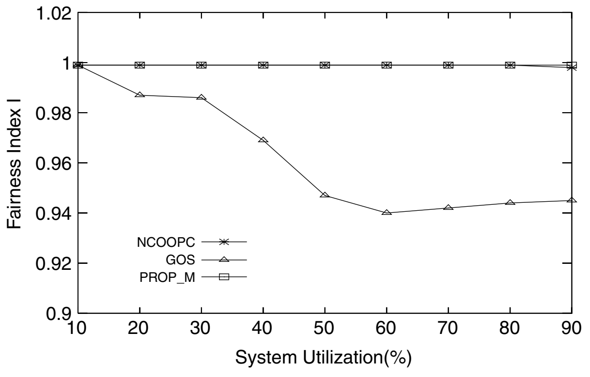
In Fig. 8(a), we present the expected response time of the system for different values of system utilization ranging from 10% to 90%. It can be observed that the performance of NCOOPC and GOS are similar for ρ ranging from 10% to 50%. NCOOPC performs significantly better than PROP_M for ρ ranging from 60% to 90%. For example, at 70% system utilization, the response time of NCOOPC is around 20% less than that of PROP_M and around 8% greater than that of GOS. The poor performance of PROP_M is due to the fact that it significantly overloads the less powerful computers where as NCOOPC does not. The decentralization and stability of allocation under non-cooperative behavior are the main advantages of NCOOPC scheme.

Fig. 8(b) shows the fairness index for different values of system utilization. The PROP_M scheme maintains a fairness index of 1 over the whole range of system loads. It can be shown that the PROP_M 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 0.94 at high load. The fairness index of NCOOPC is approximately equal to 1 and each user obtains the minimum possible expected response time for her/his own jobs (*i.e.* it is user-optimal).

Fig. 9 shows the expected response time for each user considering all the schemes at medium system load ($\rho = 60\%$).



(a) Expected response time.



(b) Fairness index.

Fig. 8. Effect of system utilization.

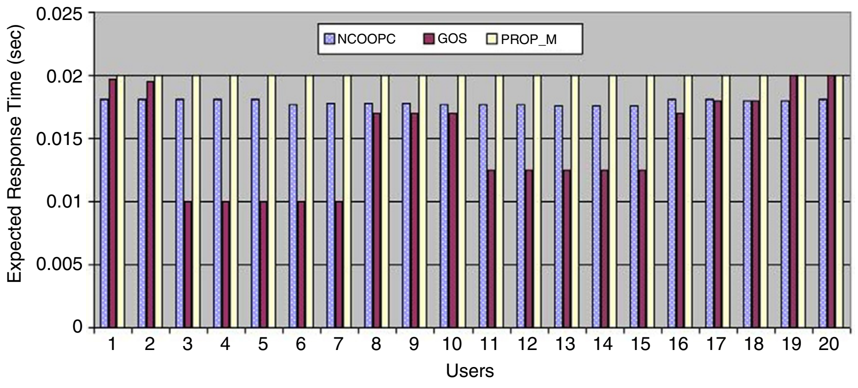
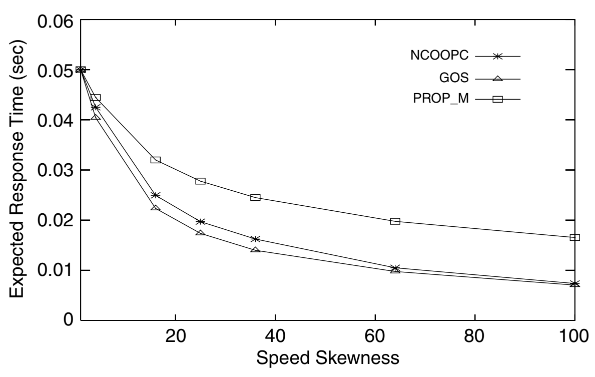
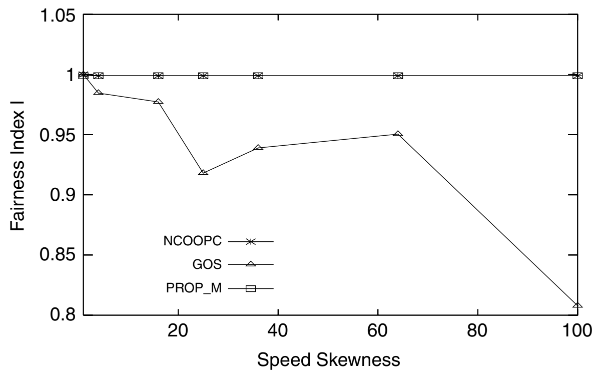


Fig. 9. Expected Response Time for each User ($\rho = 60\%$).



(a) Expected response time.



(b) Fairness index.

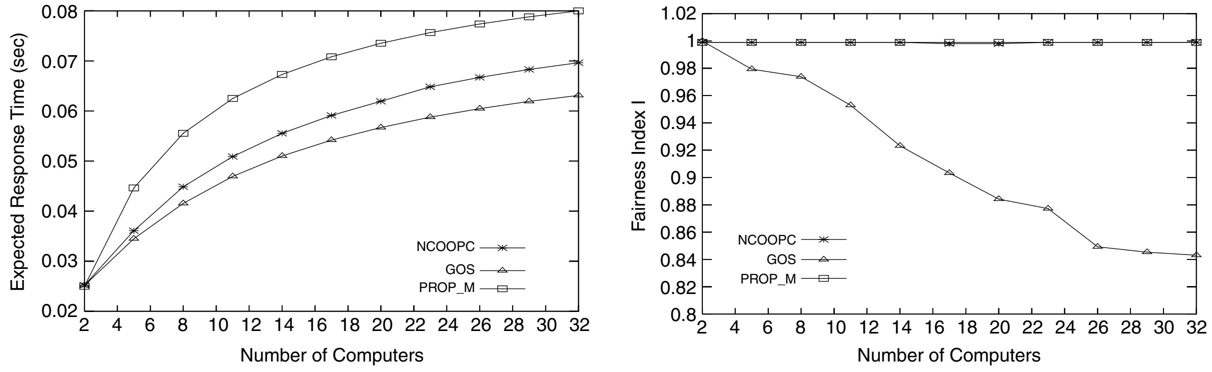
Fig. 10. Effect of heterogeneity.

The PROP_M scheme guarantees equal expected response times for all the users. The expected response times of the users in case of NCOOPC are almost the same. However, the disadvantage of PROP_M is that the users have a higher expected response time for their jobs where as NCOOPC provides the minimum possible expected execution time for each user according to the properties of the Nash equilibrium. It can be observed that in the case of GOS, there are large differences in the users' expected response times. Thus, the performance of NCOOPC is not only close to that of GOS but also makes an allocation that provides fairness to the users.

5.2. Effect of heterogeneity

To study the effect of heterogeneity on the performance of load balancing schemes, we simulated heterogeneous systems of 32 computers with configurations given in Table 4. The systems have jobs from 20 users. The system utilization was kept constant ($\rho = 60\%$) and the mean communication time t is assumed to be 0.001 s. The total arrival rates (Φ) are calculated using Eq. (31) and the arrival rates of each user (ϕ^j's) are calculated using the fractions given in Table 8. The job arrival rates of each user to each computer (ϕ_i^j's) are obtained similar to Table 5.

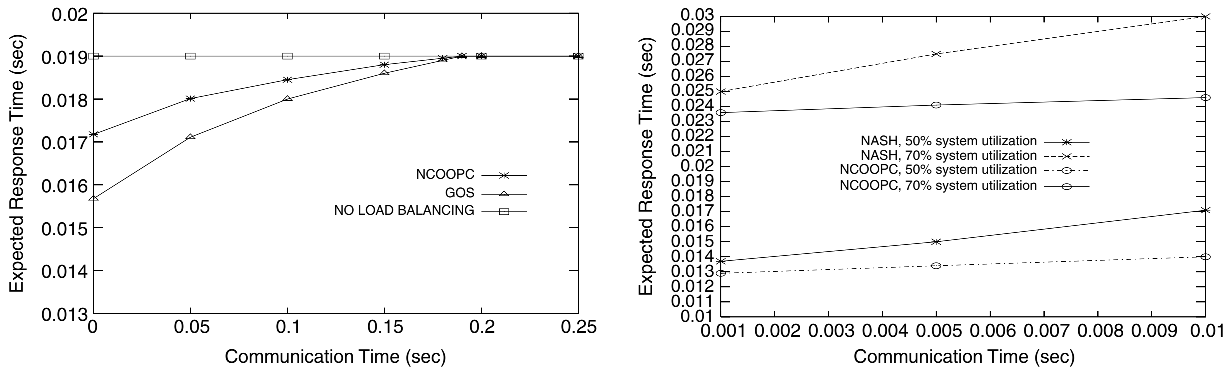
Fig. 10(a) shows the effect of speed skewness on the expected response time of all the schemes. It can be observed that as the skewness increases, the performance of NCOOPC approaches to that of GOS which means that in highly heterogeneous systems NCOOPC is very effective. PROP_M performs poorly because it overloads the slowest computers. From Fig. 10(b) it can be observed that NCOOPC maintains a fairness index of almost 1 with increasing speed skewness. The fairness index of GOS drops



(a) Expected response time.

(b) Fairness index.

Fig. 11. Effect of system size.



(a) NCOOPC.

(b) NASH vs. NCOOPC.

Fig. 12. Effect of communication time.

from 1 at low skewness to 0.81 at high skewness. This means that the GOS produces an allocation which does not guarantee equal expected response time for all the users in the system. PROP_M has a fairness index of 1 for any speed skewness. NCOOPC and PROP_M guarantees almost equal expected response times for all the users. However, the expected response times of the users in the case of NCOOPC are considerably less than that of PROP_M. The distributed nature, user-optimality, and near-optimal performance are the advantages of NCOOPC which are very important in distributed computer systems.

5.3. *Effect of system size*

In this section, we study the effect of system size on the performance of load balancing schemes. We increased the size of the heterogeneous system from 2 to 32 with configurations similar to those discussed in Section 4.3. The system utilization was kept constant ($\rho = 60\%$) and t is assumed to be 0.001 s. The total arrival rates (Φ) are calculated using Eq. (31) and the arrival rates of each user (ϕ^j's) are calculated using the fractions given in Table 8. The job arrival rates of each user to each computer (ϕ_i^j's) are obtained similar to Table 7.

Fig. 11(a) shows the expected response time where the number of computers increases from 2 to 32. It can be observed that the performance of NCOOPC lies in between GOS and PROP_M and the sub-linear increase in the expected response time of NCOOPC also shows that it is scalable. From Fig. 11(b) it can be observed that the fairness index of PROP_M is 1 and the fairness index of NCOOPC is very close to 1 with an increase in the system size. The fairness index of GOS drops considerably with an increase in the system size. The expected response times of all the users are the same in the case of PROP_M and are almost equal in the case of NCOOPC. However, the expected response times of the users are considerably less in the case of NCOOPC compared to PROP_M. Thus, NCOOPC provides an allocation which is fair to all the users independent of the allocated computers and also performs near the optimal scheme.

5.4. *Effect of communication time*

Fig. 12(a) shows the effect of increasing communication time on the performance of NCOOPC and GOS. It can be observed that, as t increases, the expected response time of NCOOPC and GOS increases and reaches 0.019 s at around $t = 0.17$ s, which is the limiting case of load balancing when the communication time is high. This is because, as t increases, the load transfer from slow computers to fast computers decreases and NCOOPC will not be able to load balance to its optimum. Similarly, the Fairness Index of NCOOPC decreases as the communication time increases.

We also implemented NASH scheme [18] for comparison. NASH is a non-cooperative game based load balancing scheme that provides fairness to the users. However, it does not take the communication costs into account in finding the optimal solution. Fig. 12(b) shows the effect of communication time on the performance of NASH and NCOOPC at 50% and 70% system utilizations for the configuration given in Table 1. It can be observed that, as t increases, the performance degradation of NASH is higher compared to that of NCOOPC. For example, as t increases from 0.001 to 0.01 s, the performance degradation of NASH is around 25% whereas the performance of NCOOPC degrades by only around 5%. This is because NCOOPC takes the communication delays into account whereas NASH does not. Thus, NCOOPC can show a significant performance improvement over NASH.

6. Conclusions and future work

In this paper, we proposed two fair load balancing schemes for distributed systems by taking the communication costs into account. Using cooperative game theory we proposed the CCOOP algorithm that provides fairness to all the jobs in a single-class job distributed system. Using non-cooperative game theory we proposed the NCOOPC distributed algorithm that provides fairness to all users in a multi-user job distributed system. The derivation of CCOOP and NCOOPC is validated theoretically using Game Theory results and their performance is evaluated using simulations. The simulations were performed on a variety of system configurations that allowed us to compare the schemes in a fair manner. The experimental results showed that CCOOP and NCOOPC not only perform near the system optimal schemes OPTIM and GOS respectively but also provide fairness to the users and their jobs. CCOOP and NCOOPC will be suitable for systems in which the fair treatment of the users or their jobs is as important as other performance characteristics.

In future work, we plan to implement the proposed schemes on real distributed systems consisting of heterogeneous computers in order to validate our results and develop mechanisms that take into account the selfish behavior of the entities in the system. We also plan to develop dynamic load balancing schemes based on dynamic game theory that provide fairness by taking the current system load into account and also consider other aspects of heterogeneity.

Acknowledgments

We wish to express our sincere thanks to the reviewers for their helpful and constructive suggestions which considerably improved the quality of the manuscript.

Appendix A

For the cooperative load balancing game defined in Definition 2.1, we are interested in finding the NBS which provides a Pareto optimal solution. We use the existing theory on cooperative games [34,36,47,52,14] (and references there-in) in the proofs of Theorems 2.1 and 2.2.

Definition A.1 (*Pareto Optimality*). Let $U \subset \mathbf{R}^N$ be the set of achievable performances. The point $u \in U$ is said to be Pareto optimal if for each $v \in U$, $v \le u$, then $v = u$.

Definition A.2 (*The Nash Bargaining Solution (NBS)*). A mapping $S : G \to \mathbf{R}^N$ (where G denotes the set of achievable performances with respect to the initial agreement point) is said to be a NBS if: (a) $S(U, \mathbf{u}^0) \in U_0$; (b) $S(U, \mathbf{u}^0)$ is Pareto optimal and satisfies the fairness axioms.

Definition A.3 (*Bargaining Point*). $\mathbf{u}^*$ is a *(Nash) bargaining point* if it is given by $S(U, \mathbf{u}^0)$ and $\mathbf{f}^{-1}(\mathbf{u}^*)$ is called the set of *(Nash) bargaining solutions*.

Theorem A.1 (*Nash Bargaining Point Characterization [47,52]*). *Consider the assumptions from Definitions A.1–A.3 above. Let J denote the set of players who are able to achieve a performance strictly superior to their initial performance and let X_0 denote the set of strategies that enable the players to achieve at least their initial performances. Let the vector function $\{f_j\}, j \in J$ be one-to-one on X_0. Then, there exists a unique bargaining point $\mathbf{u}^*$ and the set of the bargaining solutions $\mathbf{f}^{-1}(\mathbf{u}^*)$ is determined by solving the following optimization problems:*

$$(P_J) : \min_{\mathbf{x}} \prod_{j\in J} (f_j(\mathbf{x}) - u_j^0) \quad \mathbf{x} \in X_0 \tag{35}$$

$$(P'_J) : \min_{\mathbf{x}} \sum_{j\in J} \ln(f_j(\mathbf{x}) - u_j^0) \quad \mathbf{x} \in X_0. \tag{36}$$

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

Proof of Theorem 2.1. The objective function for each player $f_i(X)$ (Definition 2.1) is convex and bounded below. The set of constraints is convex and compact. Thus, the conditions in Theorem A.1 are satisfied and the result follows. □

Proof of Theorem 2.2. The objective vector function $\{f_j\}, j \in 1, \ldots, n+1$ (Definition 2.1) of the players is a one-to-one function of X. Thus, applying Theorem A.1 the result follows. □

Proof of Theorem 2.3. Let u_i and v_i denote the network traffic into node i and the network traffic out of node i respectively. From the balance of the total traffic in the network, we have

$$\sum_{i=1}^{n} u_i = \sum_{i=1}^{n} v_i. \tag{37}$$

The load β_i on node i can then be written as

$$\beta_i = \phi_i + u_i - v_i \tag{38}$$

and the network traffic λ can be written as

$$\lambda = \sum_{i=1}^{n} v_i \quad \left(= \sum_{i=1}^{n} u_i\right). \tag{39}$$

Hence, the problem becomes:

$$\min E(u, v) = \left[\sum_{i=1}^{n} \ln D_i(\phi_i + u_i - v_i) + \ln G\left(\sum_{i=1}^{n} v_i\right)\right] \tag{40}$$

subject to

$$\beta_i = \phi_i + u_i - v_i \ge 0, \quad i = 1, \ldots, n \tag{41}$$

$$-\sum_{i=1}^{n} u_i + \sum_{i=1}^{n} v_i = 0 \tag{42}$$

$$\beta_i = \phi_i + u_i - v_i < \mu_i, \quad i = 1, \ldots, n \tag{43}$$

$$u_i \ge 0, \quad i = 1, \ldots, n \tag{44}$$

$$v_i \ge 0, \quad i = 1, \ldots, n. \tag{45}$$

The objective function in Eq. (40) is convex and the constraints are all linear and define a convex polyhedron. This imply that the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality [42].

Let $\alpha, \delta_i \le 0, \eta_i \le 0, \psi_i \le 0$ denote the Lagrange multipliers [42]. The Lagrangian is

$$L(u, v, \alpha, \delta, \eta, \psi) = E(u, v) + \alpha\left(-\sum_{i=1}^{n} u_i + \sum_{i=1}^{n} v_i\right) + \sum_{i=1}^{n} \delta_i(\phi_i + u_i - v_i) + \sum_{i=1}^{n} \eta_i u_i + \sum_{i=1}^{n} \psi_i v_i. \tag{46}$$

We ignore the constraint in Eq. (43) since all the associated multipliers will be zero if we introduce it in the Lagrangian. The optimal solution satisfies the following Kuhn–Tucker conditions:

$$\frac{\partial L}{\partial u_i} = d_i(\phi_i + u_i - v_i) - \alpha + \delta_i + \eta_i = 0, \quad i = 1, \ldots, n \tag{47}$$

$$\frac{\partial L}{\partial v_i} = -d_i(\phi_i + u_i - v_i) + g\left(\sum_{i=1}^{n} v_i\right) + \alpha - \delta_i + \psi_i = 0, \quad i = 1, \ldots, n \tag{48}$$

$$\frac{\partial L}{\partial \alpha} = -\sum_{i=1}^{n} u_i + \sum_{i=1}^{n} v_i = 0 \tag{49}$$

$$\begin{aligned} &\phi_i + u_i - v_i \geq 0, \\ &\delta_i(\phi_i + u_i - v_i) = 0, \quad \delta_i \leq 0,\ i = 1, \ldots, n \end{aligned} \tag{50}$$

$$u_i \geq 0, \quad \eta_i u_i = 0, \quad \eta_i \leq 0,\ i = 1, \ldots, n \tag{51}$$

$$v_i \geq 0, \quad \psi_i v_i = 0, \quad \psi_i \leq 0,\ i = 1, \ldots, n. \tag{52}$$

In the following, we find an equivalent form of Eqs. (47)–(52) in terms of β_i. Adding Eqs. (47) and (48) we have, $-g(\sum v_i) = \eta_i + \psi_i,\ i = 1, \ldots, n$. Since $g > 0$, either $\eta_i < 0$ or $\psi_i < 0$ (or both). Hence, from Eqs. (51) and (52), for each i, either $u_i = 0$ or $v_i = 0$ (or both). We consider each case separately.

- *Case* I: $u_i = 0, v_i = 0$: Then, we have $\beta_i = \phi_i$. It follows from Eq. (50) that $\delta_i = 0$. Substituting this into Eqs. (47) and (48) gives

$$d_i(\beta_i) = \alpha - \eta_i \geq \alpha \tag{53}$$

$$d_i(\beta_i) = \alpha + g(\lambda) + \psi_i \leq \alpha + g(\lambda). \tag{54}$$

From the above, we have

$$\alpha \leq d_i(\beta_i) \leq \alpha + g(\lambda). \tag{55}$$

This case corresponds to neutral nodes.

- *Case* II: $u_i = 0, v_i > 0$: Then, from Eq. (52), we have $\psi_i = 0$. We consider the following subcases.
  - *Case* II.1 $v_i < \phi_i$: Then, we have $0 < \beta_i < \phi_i$. It follows from Eq. (50) that $\delta_i = 0$. Substituting this into Eqs. (47) and (48) gives

$$d_i(\beta_i) = \alpha - \eta_i \geq \alpha \tag{56}$$

$$d_i(\beta_i) = g(\lambda) + \alpha. \tag{57}$$

  This case corresponds to active sources.
  - *Case* II.2 $v_i = \phi_i$: Then, we have $\beta_i = 0$ and Eqs. (47) and (48) gives

$$d_i(\beta_i) = \alpha - \delta_i - \eta_i \geq \alpha \tag{58}$$

$$d_i(\beta_i) = \alpha + g(\lambda) - \delta_i \geq \alpha + g(\lambda). \tag{59}$$

  Thus, we have

$$d_i(\beta_i) \geq \alpha + g(\lambda). \tag{60}$$

  This case corresponds to idle sources.
- *Case* III: $u_i > 0, v_i = 0$: Then, we have $\beta_i > \phi_i$. It follows from Eqs. (50) and (51) that $\delta_i = 0$ and $\eta_i = 0$. Substituting this into Eq. (47), we have

$$d_i(\beta_i) = \alpha. \tag{61}$$

This case corresponds to sinks.

Eq. (49) may be written in terms of β_i as

$$\sum_{i=1}^{n} \beta_i = \Phi. \tag{62}$$

Using Eqs. (57) and (61), the above equation becomes

$$\sum_{i \in S} d_i^{-1}(\alpha) + \sum_{i \in N} \phi_i + \sum_{i \in R_a} d_i^{-1}(\alpha + g(\lambda)) = \Phi \tag{63}$$

which is the total flow constraint. □

Definition A.4. From Theorem 2.3, the following properties which are true in the optimal solution can be derived and their proofs are similar to those in [26]. The conditions in Property A.1 help to partition the nodes into one of the four categories. Once the node partition is known, the optimal loads for each node can be calculated based on Property A.2. Property A.3 states that the job flow out of the sources equals the job flow into the sinks.

Property A.1.

$$d_i(0) \geq \alpha + g(\lambda), \quad \textit{iff} \quad \beta_i = 0 \tag{64}$$

$$d_i(\phi_i) > \alpha + g(\lambda) > d_i(0), \quad \textit{iff} \quad 0 < \beta_i < \phi_i \tag{65}$$

$$\alpha \leq d_i(\phi_i) \leq \alpha + g(\lambda), \quad \textit{iff} \quad \beta_i = \phi_i \tag{66}$$

$$\alpha > d_i(\phi_i), \quad \textit{iff} \quad \beta_i > \phi_i. \tag{67}$$

Remark A.1. Property A.1 states that at an optimal solution (α): the differential node delay of a sink node (α) is greater than its initial (prior to load balancing) differential node delay (Eq. (67)); the differential node delay of an active source node lies between its initial differential node delay and the differential node delay when the node has no load (this delay consists of the differential node delay at a sink and the differential communication delay due to sending a job through the network to a sink) (Eq. (65)); the differential node delay of a neutral node is the same as its initial differential node delay and lies between that of a sink and an active source (Eq. (66)); and the differential node delay of an idle source node is not less than the differential node delay of an active source (Eq. (64)).

Property A.2. *If β is an optimal solution to the problem in Theorem* 2.2*, then we have:*

$$\beta_i = 0, \quad i \in R_d \tag{68}$$

$$\beta_i = d_i^{-1}(\alpha + g(\lambda)), \quad i \in R_a \tag{69}$$

$$\beta_i = \phi_i, \quad i \in N \tag{70}$$

$$\beta_i = d_i^{-1}(\alpha), \quad i \in S. \tag{71}$$

Remark A.2. Property A.2 states that the optimal load of an idle source node is 0 (Eq. (68)); the optimal load of a neutral node is the same as its initial load (job arrival rate) (Eq. (70)); and the optimal loads of an active source node and a sink node are given by their inverse (functions) differential node delays (Eqs. (69) and (71)).

Property A.3. *If β is an optimal solution to the problem in Theorem* 2.2*, then we have $\lambda = \lambda_S = \lambda_R$, where $\lambda_S = \sum_{i \in S}[d_i^{-1}(\alpha) - \phi_i]$ and $\lambda_R = \sum_{i \in R_d} \phi_i + \sum_{i \in R_a}[\phi_i - d_i^{-1}(\alpha + g(\lambda_S))]$.* □

Remark A.3. Property A.3 states that, at an optimal solution, the total job traffic through the network is the same as the traffic from all the source nodes which is the same as the traffic to all the sink nodes.
Based on the above properties, we have the following definitions for an arbitrary α ($\geq$0):

$$S(\alpha) = \{i | d_i(\phi_i) < \alpha\} \tag{72}$$

$$\lambda_S(\alpha) = \sum_{i \in S(\alpha)} [d_i^{-1}(\alpha) - \phi_i] \tag{73}$$

$$R_d(\alpha) = \{i | d_i(0) \geq \alpha + g(\lambda_S(\alpha))\} \tag{74}$$

$$R_a(\alpha) = \{i | d_i(\phi_i) > \alpha + g(\lambda_S(\alpha)) > d_i(0)\} \tag{75}$$

$$\lambda_R(\alpha) = \sum_{i \in R_d(\alpha)} \phi_i + \sum_{i \in R_a(\alpha)} [\phi_i - d_i^{-1}(\alpha + g(\lambda_S(\alpha)))] \tag{76}$$

$$N(\alpha) = \{i | \alpha \leq d_i(\phi_i) \leq \alpha + g(\lambda_S(\alpha))\}. \tag{77}$$

Eq. (72) denotes the set of sink nodes, Eq. (73) denotes the job traffic into the sinks, Eq. (74) denotes the set of idle source nodes, Eq. (75) denotes the set of active source nodes, Eq. (76) denotes the job traffic from the sources, and Eq. (77) denotes the set of neutral nodes.

Thus, if an optimal α is given, the node partitions in the optimal solution are characterized as $R_d = R_d(\alpha), R_a = R_a(\alpha), N = N(\alpha), S = S(\alpha)$ and $\lambda = \lambda_S = \lambda_R = \lambda_S(\alpha) = \lambda_R(\alpha)$. □

Appendix B

Proof of Theorem 3.1. We restate the problem introducing the variables u_i^j and v_i^j which denote the user j's network traffic into node i and network traffic out of node i respectively. From the balance of the total traffic of user j in the network, we have

$$\lambda^j = \sum_{i=1}^n u_i^j = \sum_{i=1}^n v_i^j. \tag{78}$$

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

$$\beta_i^j = \phi_i^j + u_i^j - v_i^j, \quad i = 1, \ldots, n. \tag{79}$$

Using the above equations, the problem in Eq. (19) becomes

$$\min_{u_i^j, v_i^j} D^j(u, v)$$
$$= \left[\frac{1}{\phi^j} \sum_{i=1}^n \frac{(\phi_i^j + u_i^j - v_i^j)}{(\mu_i^j - (\phi_i^j + u_i^j - v_i^j))} + \frac{t \sum_{i=1}^n v_i^j}{\phi^j \left(g_{-j} - t \sum_{i=1}^n v_i^j \right)} \right] \tag{80}$$

subject to the following constraints:

$$-\sum_{i=1}^n u_i^j + \sum_{i=1}^n v_i^j = 0 \tag{81}$$

$$u_i^j - v_i^j + \phi_i^j \geq 0, \quad i = 1, \ldots, n \tag{82}$$

$$u_i^j \geq 0, \quad i = 1, \ldots, n \tag{83}$$

$$v_i^j \geq 0, \quad i = 1, \ldots, n. \tag{84}$$

We ignore the stability constraint (Eq. (22)) because at Nash equilibrium it is always satisfied and the total arrival rate (Φ) does not exceed the total processing rate of the system. The objective function in Eq. (80) is convex and the constraints are all linear. This implies that the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality [42]. The total arrival rate of user j (ϕ^j) is constant.

Let $\alpha^j, \delta_i \leq 0, \psi_i \leq 0, \eta_i \leq 0$, denote the Lagrange multipliers [42]. The Lagrangian is

$$L(u^j, v^j, \alpha^j, \delta, \psi, \eta) = \phi^j D^j(u_i^j, v_i^j) + \alpha^j \left(\sum_{i=1}^n v_i^j - \sum_{i=1}^n u_i^j \right)$$
$$+ \sum_{i=1}^n \delta_i (u_i^j - v_i^j + \phi_i^j) + \sum_{i=1}^n \psi_i u_i^j + \sum_{i=1}^n \eta_i v_i^j. \tag{85}$$

The optimal solution satisfies the following Kuhn–Tucker conditions:

$$\frac{\partial L}{\partial u_i^j} = f_i^j(\phi_i + u_i - v_i) - \alpha^j + \delta_i + \psi_i = 0, \quad i = 1, \ldots, n. \tag{86}$$

$$\frac{\partial L}{\partial v_i^j} = -f_i^j(\phi_i + u_i - v_i) + g^j\left(\sum_{l=1}^n v_l \right) + \alpha^j - \delta_i + \eta_i = 0, \quad i = 1, \ldots, n. \tag{87}$$

$$-\sum_{i=1}^n u_i^j + \sum_{i=1}^n v_i^j = 0 \tag{88}$$

$$\phi_i^j + u_i^j - v_i^j \geq 0,$$
$$\delta_i(\phi_i^j + u_i^j - v_i^j) = 0, \quad \delta_i \leq 0, \ i = 1, \ldots, n. \tag{89}$$

$$u_i^j \geq 0, \qquad \psi_i u_i^j = 0, \quad \psi_i \leq 0, \ i = 1, \ldots, n. \tag{90}$$

$$v_i^j \geq 0, \qquad \eta_i v_i^j = 0, \quad \eta_i \leq 0, \ i = 1, \ldots, n. \tag{91}$$

In the following, we find an equivalent form of Eqs. (86)–(91) in terms of β_i. We consider two cases:

- *Case* I: $u_i^j - v_i^j + \phi_i^j = 0$: From Eq. (79), we have $\beta_i^j = 0$ and since $\phi_i^j > 0$, it follows that $v_i^j > 0$. Eq. (91) implies $\eta_i = 0$. Then from Eqs. (87) and (89), we get $f_i^j(\beta_i) = \alpha^j + g^j(\lambda) - \delta_i \geq \alpha^j + g^j(\lambda)$. This case corresponds to idle sources.
- *Case* II: $u_i^j - v_i^j + \phi_i^j > 0$: From Eq. (89), we have $\delta_i = 0$.
  - *Case* II.1: $v_i^j > 0$: Then, $0 < \beta_i^j < \phi_j^i$ and from Eq. (91) we have $\eta_i = 0$. Eq. (87) implies,
    $$f_i^j(\beta_i) = \alpha^j + g^j(\lambda). \tag{92}$$
    This case corresponds to active sources.
  - *Case* II.2: $v_i^j = 0$:

    Case II.2.1: $u_i^j = 0$: Then, $\beta_i^j = \phi_i^j$.
    From Eqs. (86) and (90), we have $f_i^j(\beta_i) = \alpha^j - \psi_i \geq \alpha_j$.
    From Eqs. (87) and (91), we have $f_i^j(\beta_i) = \alpha^j + g^j(\lambda) + \eta_i \leq \alpha^j + g^j(\lambda)$. This case corresponds to neutral nodes.

    Case II.2.2: $u_i^j > 0$: Then, $\beta_i^j > \phi_i^j$.
    From Eq. (90), we have $\psi_i = 0$. Substituting this in Eq. (86), we have
    $$f_i^j(\beta_i) = \alpha^j. \tag{93}$$
    This case corresponds to sink nodes.

Eq. (88) may be written in terms of β_i^j as $\sum_{i=1}^n \beta_i^j = \phi_i^j$ and using the Eqs. (92) and (93), this can be written as

$$\sum_{i \in S^j} (f_i^j)^{-1}(\beta_i|_{\beta_i^j = \alpha^j}) + \sum_{i \in N^j} \phi_i^j + \sum_{i \in R_a^j} (f_i^j)^{-1}(\beta_i|_{\beta_i^j = \alpha^j + g^j(\lambda)}) = \phi^j \tag{94}$$

which is the total flow constraint for user j. □

Definition B.1. From Theorem 3.1, the following properties which are true in the optimal solution can be derived and their proofs are similar to those in [25]. The conditions in Property B.1 help to partition the nodes into one of the four categories for user j. Once the node partition for user j is known, her/his optimal loads can be calculated based on Property B.2. Property B.3 states that the job flow out of the sources equals the job flow into the sinks for each user j.

Property B.1.

$$f_i^j(\beta_i|_{\beta_i^j=0}) \geq \alpha^j + g^j(\lambda), \quad \textit{iff} \quad \beta_i^j = 0 \tag{95}$$

$$f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}) > \alpha^j + g^j(\lambda) > f_i^j(\beta_i|_{\beta_i^j=0}), \quad \textit{iff} \quad 0 < \beta_i^j < \phi_i^j \tag{96}$$

$$\alpha^j \leq f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}) \leq \alpha^j + g^j(\lambda), \quad \textit{iff} \quad \beta_i^j = \phi_i^j \tag{97}$$

$$\alpha^j > f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}), \quad \textit{iff} \quad \beta_i^j > \phi_i^j. \tag{98}$$

Remark B.1. Property B.1 states that at an optimal solution (α^j), for each user: the differential node delay of a sink node (α) is greater than its initial (prior to load balancing) differential node delay (Eq. (98)); the differential node delay of an active source node lies between its initial differential node delay and the differential node delay when the node has no load (this delay consists of the differential node delay at a sink and the differential communication delay due to sending a job through the network to a sink) (Eq. (96)); the differential node delay of a neutral node is the same as its initial differential node delay and lies between that of a sink and an active source (Eq. (97)); and the differential node delay of an idle source node is not less than the differential node delay of an active source (Eq. (95)).

Property B.2. *If β^j is an optimal solution to the problem in Eq. (19), then we have:*

$$\beta_i^j = 0, \quad i \in R_d^j \tag{99}$$

$$\beta_i^j = (f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j+g^j(\lambda)}), \quad i \in R_a^j \tag{100}$$

$$\beta_i^j = \phi_i^j, \quad i \in N^j \tag{101}$$

$$\beta_i^j = (f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j}), \quad i \in S^j. \tag{102}$$

Remark B.2. Property B.2 states that for each user, the optimal load of an idle source node is 0 (Eq. (99)); the optimal load of a neutral node is the same as its initial load (job arrival rate) (Eq. (101)); and the optimal loads of an active source node and a sink node are given by their inverse (functions) differential node delays (Eqs. (100) and (102)).

Property B.3. *If β^j is an optimal solution to the problem in Eq. (19), then we have $\lambda^j = \lambda_S^j = \lambda_R^j$, where $\lambda_S^j = \sum_{i\in S^j}[(f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j}) - \phi_i^j]$ and $\lambda_R^j = \sum_{i\in R_d^j}\phi_i^j + \sum_{i\in R_a^j}[\phi_i^j - (f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j+g^j(\lambda_S)})]$.* $\square$

Remark B.3. Property B.3 states that, at an optimal solution, the total job traffic through the network of user *j* is the same as the traffic from all the source nodes of user *j* which is the same as the traffic to all the sink nodes of user *j*.
Based on the above properties, we have the following definitions for an arbitrary α^j (≥ 0):

$$S^j(\alpha^j) = \{i|f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}) < \alpha^j\} \tag{103}$$

$$\lambda_S^j(\alpha^j) = \sum_{i\in S^j(\alpha^j)}[(f_i^j)^{-1}(\beta_i|_{\beta_i^j=\alpha^j}) - \phi_i^j] \tag{104}$$

$$R_d^j(\alpha^j) = \{i|f_i^j(\beta_i|_{\beta_i^j=0}) \geq \alpha^j + g^j(\lambda|_{\lambda^j=\lambda_S^j(\alpha^j)})\} \tag{105}$$

$$\begin{aligned} R_a^j(\alpha^j) &= \{i|f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}) > \alpha^j + g^j(\lambda|_{\lambda^j=\lambda_S^j(\alpha^j)}) \\ &\quad > f_i^j(\beta_i|_{\beta_i^j=0})\} \end{aligned} \tag{106}$$

$$\begin{aligned} \lambda_R^j(\alpha^j) &= \sum_{i\in R_d^j(\alpha^j)}\phi_i^j + \sum_{i\in R_a^j(\alpha^j)}[\phi_i^j - (f_i^j)^{-1} \\ &\quad \times (\beta_i|_{\beta_i^j=\alpha^j+g^j(\lambda|_{\lambda^j=\lambda_S^j(\alpha^j)})})] \end{aligned} \tag{107}$$

$$N^j(\alpha^j) = \{i|\alpha^j \leq f_i^j(\beta_i|_{\beta_i^j=\phi_i^j}) \leq \alpha^j + g^j(\lambda|_{\lambda^j=\lambda_S^j(\alpha^j)})\}. \tag{108}$$

Eq. (103) denotes the set of sink nodes for user *j*, Eq. (104) denotes the job traffic into the sinks for user *j*, Eq. (105) denotes the set of idle source nodes for user *j*, Eq. (106) denotes the set of active source nodes for user *j*, Eq. (107) denotes the job traffic from the sources for user *j*, and Eq. (108) denotes the set of neutral nodes for user *j*.
Thus, if an optimal α^j is given, the node partitions in the optimal solution are characterized as $R_d^j = R_d^j(\alpha^j)$, $R_a^j = R_a^j(\alpha^j)$, $N^j = N^j(\alpha^j)$, $S^j = S^j(\alpha^j)$ and $\lambda^j = \lambda_S^j = \lambda_R^j = \lambda_S^j(\alpha^j) = \lambda_R^j(\alpha^j)$. $\square$

References

[1] I. Ahmad, A. Ghafoor, Semidistributed load balancing for massively parallel multicomputer systems, IEEE Trans. Softw. Eng. 17 (10) (1991) 987–1004.
[2] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, Adv. Comput. 63 (2005) 91–128.
[3] E. Altman, T. Basar, T. Jimenez, N. Shimkin, Routing in two parallel links: game-theoretic distributed algorithms, J. Parallel Distrib. Comput. 61 (9) (2001) 1367–1381.
[4] Amazon elastic compute cloud. http://www.amazon.com/.
[5] T.E. Anderson, D.E. Culler, D.A. Patterson, The NOW team, a case for now (networks of workstations), IEEE Micro. 15 (1) (1995) 54–64.
[6] X. Bai, D.C. Marinescu, L. Boloni, H.J. Siegel, R.A. Daley, I.-J. Wang, A macroeconomic model for resource allocation in large-scale distributed systems, J. Parallel Distrib. Comput. 68 (2) (2008) 182–199.
[7] T.D. Braun, H.J. Siegel, A.A. Maciejewski, Y. Hong, Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions, J. Parallel Distrib. Comput. 68 (11) (2008) 1504–1516.
[8] Y.C. Chow, W.H. Kohler, Models for dynamic load balancing in a heterogeneous multiple processor system, IEEE Trans. Comput. C-28 (5) (1979) 354–361.
[9] R. Cubert, P. Fishwick, Sim++ reference manual, in: CISE, University of Florida, July 1995.
[10] J. Dongarra, A. Lastovetsky, An overview of heterogeneous high performance and grid computing, in: Engineering the Grid: Status and Perspective, American Scientific Publishers, 2006.
[11] P. Dubey, Inefficiency of nash equilibria, Math. Oper. Res. 11 (1) (1986) 1–8.
[12] A.A. Economides, J. Silvester, A game theory approach to cooperative and non-cooperative routing problems, in: Proc. of the Telecommunication Symp., 1990, pp. 597–601.
[13] R. Freund, H.J. Siegel, Heterogeneous processing, IEEE Comput. Mag. 26 (6) (1993) 13–17.
[14] D. Fudenberg, J. Tirole, Game Theory, The MIT Press, 1994.
[15] L. Georgiadis, C. Nikolaou, A. Thomasian, A fair workload allocation policy for heterogeneous systems, J. Parallel Distrib. Comput. 64 (4) (2004) 507–519.
[16] P. Ghosh, K. Basu, S.K. Das, A game theory-based pricing strategy to support single/multiclass job allocation schemes for bandwidth-constrained distributed computing systems, IEEE Trans. Parallel Distrib. Syst. 18 (3) (2007) 289–306.
[17] P. Ghosh, N. Roy, S.K. Das, K. Basu, A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework, J. Parallel Distrib. Comput. 65 (11) (2005) 1366–1383.
[18] D. Grosu, A.T. Chronopoulos, Noncooperative load balancing in distributed systems, J. Parallel Distrib. Comput. 65 (9) (2005) 1022–1034.
[19] D. Grosu, A.T. Chronopoulos, M.Y. Leung, Load balancing in distributed systems: an approach using cooperative games, in: Proc. of the 16th IEEE Intl. Parallel and Distributed Processing Symp., Ft Lauderdale, Florida, USA, April 2002.
[20] D. Grosu, A.T. Chronopoulos, M.Y. Leung, Cooperative load balancing in distributed systems, Concurr. Comput. Pract. Exp. 20 (16) (2008) 1953–1976.
[21] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley-Interscience, 1991.
[22] H. Kameda, J. Li, C. Kim, Y. Zhang, Optimal Load Balancing in Distributed Computer Systems, Springer Verlag, London, 1997.
[23] S.U. Khan, I. Ahmad, Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation, in: Proc. of the Intl. Parallel and Distributed Proc. Symp., Rhodes, Greece, April 2006.
[24] J.-K. Kim, D.A. Hensgen, T. Kidd, H.J. Siegel, D.S. John, C. Irvine, T. Levin, N.W. Porter, V.K. Prasanna, R.F. Freund, A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems, Cluster Comput. 9 (3) (2006) 281–296 (special issue on Cluster Computing in Science and Engineering).
[25] C. Kim, H. Kameda, Optimal static load balancing of multi-class jobs in a distributed computer system, in: Proc. of the 10th Intl. Conf. on Distributed Computing Systems, May 1990, pp. 562–569.
[26] C. Kim, H. Kameda, An algorithm for optimal static load balancing in distributed computer systems, IEEE Trans. Comput. 41 (3) (1992) 381–384.
[27] Y.A. Korilis, A.A. Lazar, A. Orda, Capacity allocation under noncooperative routing, IEEE Trans. Automat. Control 42 (3) (1997) 309–325.
[28] E. Koutsoupias, C. Papadimitriou, Worst-case equilibria, in: Proc. of the 16th Annual Symp. on Theoretical Aspects of Computer Science, 1999, pp. 404–413.
[29] Y.K. Kwok, K. Hwang, S. Song, Selfish grids: game-theoretic modeling and NAS/PSA benchmark evaluation, IEEE Trans. Parallel Distrib. Syst. 18 (5) (2007) 621–636.
[30] H. Lee, Optimal static distribution of prioritized customers to heterogeneous parallel servers, Comput. Oper. Res. 22 (10) (1995) 995–1003.
[31] J. Li, H. Kameda, A decomposition algorithm for optimal static load balancing in tree hierarchy network configuration, IEEE Trans. Parallel Distrib. Syst. 5 (5) (1994) 540–548.
[32] J. Li, H. Kameda, Optimal static load balancing in star network configurations with two-way traffic, J. Parallel Distrib. Comput. 23 (3) (1994) 364–375.
[33] J. Li, H. Kameda, Load balancing problems for multiclass jobs in distributed/parallel computer systems, IEEE Trans. Comput. 47 (3) (1998) 322–332.
[34] A. Mas-Collel, M.D. Whinston, J.R. Green, Microeconomic Theory, Oxford Univ. Press, New York, 1995.
[35] M. Mavronicolas, P. Spirakis, The price of selfish routing, in: Proc. of the 33rd Annual ACM Symp. on Theory of Computing, July 2001, pp. 510–519.
[36] A. Muthoo, Bargaining Theory with Applications, Cambridge Univ. Press, Cambridge, UK, 1999.

[37] L.M. Ni, K. Hwang, Adaptive load balancing in a multiprocessor system with many job classes, IEEE Trans. Softw. Eng. SE-11 (5) (1985) 491–496.
[38] M. Koh, On-demand computing using network.com, in: Proceedings of the International Symposium on Grid computing, April 7–11, 2008, Taipei, Taiwan.
[39] A. Orda, R. Rom, N. Shimkin, Competitive routing in multiuser communication networks, IEEE/ACM Trans. Netw. 1 (5) (1993) 510–521.
[40] S. Penmatsa, A.T. Chronopoulos, Cooperative load balancing for a network of heterogeneous computers, in: Proc. of the 20th IEEE Intl. Parallel and Distributed Processing Symposium, 15th Heterogeneous Computing Workshop, Rhodes Island, Greece, April 2006.
[41] S. Penmatsa, A.T. Chronopoulos, Price-based user-optimal job allocation scheme for grid systems, in: Proc. of the 20th IEEE Intl. Parallel and Distributed Processing Symposium, 3rd High Performance Grid Computing Workshop, Rhodes Island, Greece, April 2006.
[42] G.V. Reklaitis, A. Ravindran, K.M. Ragsdell, Engineering Optimization: Methods and Applications, Wiley-Interscience, 1983.
[43] K.W. Ross, D.D. Yao, Optimal load balancing and scheduling in a distributed computer system, J. ACM 38 (3) (1991) 676–690.
[44] T. Roughgarden, Stackelberg scheduling strategies, in: Proc. of the 33rd Annual ACM Symp. on Theory of Computing, July 2001, pp. 104–113.
[45] K. Rzadca, D. Trystram, A. Wierzbicki, Fair game-theoretic resource management in dedicated grids, in: Proc. of the 7th IEEE Intl. Symp. on Cluster Computing and the Grid, Brazil, May 2007, pp. 343–350.
[46] N.G. Shivaratri, P. Krueger, M. Singhal, Load distributing for locally distributed systems, Comput. 25 (12) (1992) 33–44.
[47] A. Stefanescu, M.V. Stefanescu, The arbitrated solution for multi-objective convex programming, Rev. Roum. Math. Pure Appl. 29 (1984) 593–598.
[48] R. Subrata, A. Zomaya, B. Landfeldt, A cooperative game framework for QoS guided job allocation schemes in grids, IEEE Trans. Comput. 57 (10) (2008) 1413–1422.
[49] R. Subrata, A. Zomaya, B. Landfeldt, Game theoretic approach for load balancing in computational grids, IEEE Trans. Parallel Distrib. Syst. 19 (1) (2008) 66–76.
[50] X. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: Proc. of the Intl. Conf. on Parallel Processing, August 2000, pp. 373–382.
[51] A.N. Tantawi, D. Towsley, Optimal static load balancing in distributed computer systems, J. ACM 32 (2) (1985) 445–465.
[52] H. Yaiche, R.R. Mazumdar, C. Rosenberg, A game theoretic framework for bandwidth allocation and pricing in broadband networks, IEEE/ACM Trans. Netw. 8 (5) (2000) 667–678.
[53] Y. Zhang, H. Kameda, S.L. Hung, Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems, IEE Proc. Comput. Digit. Tech. 144 (2) (1997) 100–106.
[54] Q. Zheng, C.-K. Tham, B. Veeravalli, Dynamic load balancing and pricing in grid computing with communication delay, J. Grid Comput. 6 (3) (2008) 239–253.

Satish Penmatsa received his M.S. and Ph.D. in Computer Science from the University of Texas at San Antonio in 2003 and 2007 respectively. He is currently with the Department of Mathematics and Computer Science at the University of Maryland Eastern Shore. His research interests are in the areas of parallel and distributed systems, high performance computing, grid computing, wireless networks, game theory, science and engineering applications. He is a member of IEEE, IEEE Computer Society, and the ACM.

Anthony T. Chronopoulos received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1987. He is currently a Professor in Computer Science at the University of Texas at San Antonio. He has published 39 journal and 51 refereed conference proceedings publications in the areas of Distributed Systems, High Performance Computing and Applications. He has been awarded 13 federal/state government research grants. His work is cited in over 220 non-coauthors' research articles. He is a senior member of IEEE (since 1997).