Future Generation Computer Systems 109 (2020) 29–44

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A game-theoretic approach for cost-aware load balancing in distributed systems

Avadh Kishor [a,*], Rajdeep Niyogi [a], Bharadwaj Veeravalli [b]

[a] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India
[b] Department of Electrical and Computer Engineering, National University of Singapore, Singapore

ARTICLE INFO

Article history:
Received 29 August 2019
Received in revised form 21 February 2020
Accepted 10 March 2020
Available online 14 March 2020

Keywords:
Distributed system
Cost minimization
Load balancing game
Nash equilibrium

ABSTRACT

In this paper, we consider a load balancing problem in distributed systems, that has two conflicting objectives: (i) minimizing the users' expected response time and (ii) minimizing the total monetary cost incurred by each user. To satisfy both the objectives simultaneously, we consider the objectives in an integrated manner and formulate the problem as an optimization problem. We then cast it into a game-theoretic setting and model the load balancing problem as a non-cooperative game. To solve the game, we characterize the best response strategy for each player, and derive a decentralized algorithm called **C**ost-**A**ware **L**oad **B**alancing **A**lgorithm (*CALBA*). We conduct a rigorous experimental study to demonstrate the effectiveness of *CALBA*. Further, to establish the effectiveness of *CALBA*, we compare it with three other load balancing schemes, i.e., *MinRT*, *MinCost*, and *GPMS*, using various system configurations such as varying system size, varying system utilization, and system heterogeneity, across multiple performance indicators. The computational results show that textitCALBA outperforms the competitive schemes by reducing the response time and cost, and unlike others, *CALBA* produces an allocation of load which guarantees fairness (in terms of response time) between the users. In a nutshell, the results demonstrate the suitability of *CALBA* in realistic scenarios as it is an adaptable and feasible approach to get a cost-aware load balancing solution.

1. Introduction

Distributed systems have received a lot of attention over the past decades [1,2]. In general, distributed systems can be considered as a collection of autonomous computing resources which are shared by a group of independent users. The performance of the distributed system is dependent on the allocation of users' jobs on computing resources. The problem of effectively managing the utilization of resources is called load balancing [3]. Thus, to make effective utilization of resources in such systems, an efficient load balancing scheme is required. Load balancing problem has been studied extensively in various contexts and there exist several variations of the problem with different names, e.g., resource allocation, task assignment, scheduling, and load sharing.

We consider a load balancing problem in a distributed system where a set of independent users are sharing a set of computing resources; henceforth referred to as servers. We assume that the servers are heterogeneous in terms of their processing capacity (i.e., service rate) speed and are owned by an individual; henceforth, referred to as service provider. The price per unit service rate of each server is fixed by the service provider, and the users are charged on per unit time basis. Each user is autonomous and self-interested. Here autonomous means a user is free to decide without any controlling authority, and self-interested means a user is interested in optimizing its utility. In this paper, we consider two conflicting objectives: (i) minimizing the users' expected response time and (ii) minimizing the total monetary cost incurred by each user.

1.1. Motivation

Several real-world systems fit into the aforementioned load balancing scenario. In particular, our study is inspired by the cloud computing systems [2] where a large number of computational devices are shared among a large number of users. The load balancing in cloud computing is using Infrastructure-as-a-service (IaaS). IaaS is a service model that offers a cluster of machines (physical or virtual) for clients to perform their tasks. In IaaS system, each machine acts as a server and the computing speed of servers may be different. The clients of the cloud are charged according to the per CPU-hour consumed. The

* Corresponding author.
E-mail addresses: akishor@cs.iitr.ac.in (A. Kishor), rajdpfec@iitr.ac.in (R. Niyogi), elebv@nus.edu.sg (B. Veeravalli).

https://doi.org/10.1016/j.future.2020.03.027
0167-739X/© 2020 Elsevier B.V. All rights reserved.

cloud resources are owned by an individual (e.g., Amazon EC2[1], HP cloud.[2] In such a distributed cloud computing system, the service provider wants to effectively manage the utilization of resources and wants to maximize its revenue without charging any extra money from its clients. This follows from the fact that cloud computing is a competitive market with several other providers. In such a scenario, the survival and the success of a service provider depend even more heavily on an appropriate load balancing policy. Particularly, the service provider needs to deal with the two conflicting objectives mentioned above.When users consider the first objective, then it would be reasonable to utilize the faster servers as much as possible. Due to this temptation towards faster servers, faster servers (slower servers) become overloaded (underutilized). As a result, a user's job can take longer to execute on a server that is heavily loaded as opposed to a lightly loaded. Since the pricing scheme is based on the per unit time use, this selfishness of the users may cause them to pay more under the heavy load condition. On the other hand, when users consider the second objective, it would be reasonable to target a slower server as it is cheaper than a faster one. Due to this temptation towards slower servers, the impact of processing capacity is overlooked and slower servers (faster servers) become overloaded (underutilized). Consequently, the total monetary cost of each user is minimized but the expected response time is maximized.

There exists a good volume of works [4–15] (more detail can be found in Section 2) attempting to solve the load balancing problem in distributed systems. In these works, the problem is modeled as a non-cooperative game where the payoff of players is jobs' execution time. Hence, all these works have considered only one objective, i.e., to minimize the users' expected response time. To the best of our knowledge, we did not find any such work that considers minimization of the expected response time of users and minimizing the monetary cost incurred by users simultaneously. In this view, we propose a load balancing scheme that, on the one hand, minimizes the expected response time of users, and, on the other hand, minimizes the total monetary cost incurred by users in executing their jobs on servers.

1.2. Our contribution

A mathematical model is given for both the above-mentioned objectives. Both the objectives are integrated to form a new objective which is defined as a penalty function of the user. We formulate the problem of minimizing the penalty function as a non-cooperative game, called non-cooperative load balancing game, between users as they compete with each other in sharing the servers. The solution concept of the game is defined as Nash equilibrium, where no player will be better off by deviating unilaterally; In other words, every player is playing the best response strategy in response to other players' strategy choices. In this view, we first characterize the best response strategy of each player and then propose an algorithm to compute this strategy. Finally, to compute the Nash equilibrium of the non-cooperative game, we propose a distributed algorithm, hereinafter, referred to as **C**ost-**A**ware **L**oad **B**alancing **A**lgorithm (*CALBA*). To test the performance of *CALBA*, we carry out an empirical study under different system loads and configurations. Moreover, we compare it against three other load balancing schemes. The empirical results show that CALBA not only provides optimal jobs execution time but also provides an optimal monetary cost for users.

The contributions of this paper are summarized as follows.

- We consider a load balancing problem in a distributed and non-cooperative environment with two conflicting objectives. We consider both the objectives in an integrated manner by applying the weighted-sum method.
- We model this problem as a non-cooperative game called non-cooperative load balancing game.
- We propose a distributed load balancing algorithm (*CALBA*) to compute the Nash equilibrium of the game.
- We perform a rigorous experimental study to show the efficacy of *CALBA*.

The rest of the paper is organized as follows. In Section 2, we present a review of the existing literature. In Section 3, we define the notations and mathematical modeling of the problem. The game formulation of the problem, the best response strategy characterization, and *CALBA* is given in Section 4. Section 5 is devoted to the experimental results and its discussions. We draw the conclusions in Section 6.

2. Literature review

Load balancing problem in distributed computing systems have been extensively studied and a good volume of literature exists (see the review papers [1,2]). In particular, the problems of optimal load distribution have been investigated by using queuing models and game theory, with various performance metrics such as expected response time, load imbalance percentage, the cost incurred by users. In this line of research, initially some researchers [10–12] have suggested centralized solutions to the load balancing problem. In practical, a centralized solution of the load balancing problem in a distributed environment is not feasible due to scalability and complexity reasons. In a distributed computing environment such as clouds interactions between different users (players) can be ignored: each players utility not only affected by her action but also by the action of the other players. Game theory can perfectly capture this inherent interaction between players. In game-theoretic settings, a natural modeling framework involves finding a stable operating point called Nash equilibrium (NE). In a similar spirit, in this paper, we suggest a game-theoretic solution to the load balancing problem. Therefore, in this section, we review the literature on approaches to solve the load balancing distributed systems, which are relevant to our work and based on the concept of non-cooperative game theory.

Game-theoretic solutions of the load balancing problem in distributed computing systems have been studied in various works. We give the description of only those studies which are closely related to our work. Grosu and Chronopoulos [4] modeled load balancing in distributed systems as a non-cooperative game. In this game, the payoff for each player is the expected response time to execute its jobs. To find the NE of the game, the authors have proposed a distributed algorithm. In [5], Kameda and Altman considered the problem of optimal flow control in communication networks and proved that in presence of self-interested users the problem of allocating the flow is similar to the tragedy of the commons problem and obtained NE is not a good solution. [7] considered the load balancing problem in distributed data centers. Subrata et al. [6] proposed a non-cooperative game-based framework to solve the load balancing problem in computational grids. In all these works, the jobs execution time of the players is considered and the server cost is overlooked completely.

Ghosh et al. [14], Yu et al. in [15], and Penmatsa and Chronopoulos [17] use a two-player bargaining model to capture the interaction between users and service providers. In this model, the service provider interacts with all the users to decide the cost and load distribution. In these works, only the cost minimization objective of the users is considered.

[1] https://aws.amazon.com/ec2/.
[2] https://www.hpe.com/.

Table 1
Comparison of our proposed scheme (*CALBA*) and other state-of-the-art load balancing schemes.

Scheme	Environment	Objective(s)	Main technique(s)
[10]	Central	Single objective (response time)	Heuristic
[11,12]	Central	Single objective (response time)	Meta-heuristic
[4–9]	Distributed	Single objective (response time)	Non-cooperative game
[13]	Distributed	Single objective (response time)	Q-learning
[14,15]	Distributed	Single objective (Cost minimization)	Non-cooperative game
[16]	Distributed	Multi-objective (response time, fair utilization)	Non-cooperative game
CALBA	Distributed	Multi-objective	Non-cooperative game

However, the load balancing problem with two objectives is considered in [16,18], and [19]. Xu and Yu in [16] considered two objectives: (i) minimizing the response time of users and (ii) minimizing the load imbalance between servers. They proposed a game-theoretic algorithm which provides fairness among users maximize the utilization of servers by minimizing the segmentation of servers. Recently, to minimize the total cost including electricity bill and monetary loss due to improper service management in geo-distributed Internet data centers, the authors in [15] presented a price-sensitivity aware load balancing scheme. They modeled the problem as a non-linear optimization problem; and proposed an iterative algorithm to solve the problem. Zheng and Veeravalli [18] proposed an aggregated approach is which combines load balancing problem with the pricing model. They proposed an algorithm that provides centralized algorithm that provides a system-wide optimal solution. Jalaparti et al. [19] proposed a market-based approach to optimize the utilities between two providers and between a user and a provider. They study how competition among the players affect their cost. In sum, they considered the cost minimization as an objective and provided a centralized solution based on the Stackelberg game.

Some other works have studied load balancing problems in different application domains. For instance, in [20], a dynamic load balancing problem to minimize the makespan and to maximize the ratio of tasks is studied. In [21] a heuristic is suggested to improve the utilization of servers by reducing the utilization skewness. Elgendy et al. [22] considered computational offloading and resource allocation problems in the mobile edge computing domain. They proposed an iterative algorithm, intending to minimize time and energy consumption. In [23], a modified bee algorithm is proposed to solve a bi-objective problem (with two conflicting objectives: minimize the experienced communication cost and response time) that arises in content delivery systems. Recently, researchers have attempted to develop the cost and energy-aware schemes to find the appropriate allocation strategies [24–26].

Table 1 shows the comparision of our proposed algorithm *CALBA* and other state-of-the-art load balancing schemes.

3. System model and problem formulation

3.1. The system model

The distributed system architecture considered in this paper, illustrated in Fig. 1, consists of three layers, including the user (client) layer, front-end node layer, and the server layer. Servers execute the requests (jobs or workload) of a user. Generally, in the cloud computing setting, to assign the workload of a user, a front-end node, called dispatcher, is needed [27]. Following the previous work [28–30], we adopt the workload distribution mechanism, which is transparent to the users. Each user will assign its workload to a dispatcher who will further distribute the workload to a set of servers chosen to accord with some load distribution policy. The server will send the processed result to the dispatchers, and then dispatchers will relay back the result to the user.

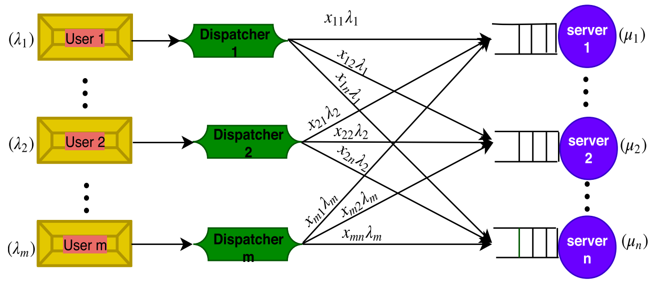


Fig. 1. The distributed system model.

Let $\mathcal{N} = \{1, 2, \ldots, n\}$ be a set of n servers owned by the service provider. The servers are heterogeneous, meaning that they have different processing capacity. The capacity of each server $j \in \mathcal{N}$ is characterized by its service rate μ_j. Each server j is associated with a price function $p_j : \mu_j \mapsto \mathbb{R}^+$. Particularly, p_j denotes the price per unit service rate of server j. We denote a set $\mathcal{M} = \{1, 2, \ldots, m\}$ of m users who are intended to share the set of n servers. Each user i has a fixed amount of workload λ_i to execute on cloud. Each user assigns its workload to a dispatcher who further distributes its workload by splitting λ_i over the set of parallel servers. Similar to [4,14], we model each server as an M/M/1-FCFS queue, which is elaborated as follows. At each server, the arrival of jobs (from dispatchers) occur according to the Poisson process, the execution times of the jobs are exponentially distributed, and jobs are executed according to first-come-first-serve (FCFS) discipline. Since all the servers are modeled as M/M/1 queue, for stability the capacity constraint of servers renders that total jobs arrival rate on a server j must be less than the processing rate of the server, i.e., $\sum_{i=1}^{m} \lambda_i < \sum_{j=1}^{n} \mu_j$.

It is important to note here that the dispatcher will take all the decisions related to the jobs' allocation. Therefore, we assume, without loss of generality, that each user is associated with a separate dispatcher. The set of users is the same as the set of dispatchers, i.e., the set of dispatchers is denoted as $\mathcal{M} = \{1, 2, \ldots, m\}$. For brevity, we henceforth shall make all the discussion regarding the dispatchers instead of users.

3.2. Load allocation model

We are given a set of dispatchers and a set of servers working independently. Each dispatcher i has a job of size λ_i and wants to assign its job by splitting it through the servers. A dispatcher can decide how its job is split among the servers, i.e., dispatcher i decides what fraction of λ_i should be assigned to each server. We denote by x_{ij} the fraction of workload of dispatcher i assigned to server j. Thus, a dispatcher i can choose any value of x_{ij}, as along as $0 \leq x_{ij} \leq 1$ and $\sum_{j=1}^{n} x_{ij} = 1$. After deciding the value of x_{ij}, dispatcher i assigns a workload to server j at the rate of $x_{ij}\lambda_i$.

The workload splitting configuration $\mathbf{x}_i$ of dispatcher i is the vector $\mathbf{x_i} = (x_{i1}, \ldots, x_{in})$, which we refer as the strategy of dispatcher i. Joint strategy profile $\mathcal{X}$ is the vector of dispatchers' strategies $\mathcal{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$. Further, in the rest of the paper, we denote: $\mathcal{X} = (\mathbf{x}_i, \mathbf{x}_{-i})$, where $\mathbf{x}_{-i}$ represents the strategy of all other dispatchers except dispatcher i.

Definition 1 (*Feasible Strategy Profile*). A strategy profile $\mathcal{X} = (\mathbf{x}_1, \dots, \mathbf{x}_m)$ is said to be a feasible strategy profile if it satisfies the following conditions:

- *Condition 1*: A dispatcher assigns non-negative amount of load to a server, i.e., we have: $x_{ij} \geq 0,\ \forall i \in \mathcal{M}, \forall j \in \mathcal{N}$.
- *Condition 2*: A dispatcher assigns its load to at least one server, i.e., we have: $\sum_{j=1}^{n} x_{ij} = 1,\ \forall i \in \mathcal{M}$.
- *Condition 3*: The total job arrival rate to a server is less than its service rate, i.e., we have: $\sum_{i=1}^{m} x_{ij}\lambda_i < \mu_j,\ \forall j \in \mathcal{N}$.

Given the joint strategy profile $\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_{-i})$, the expected response time at server j is expressed as

$$\mathcal{D}_j(\mathbf{x}_i, \mathbf{x}_{-i}) = \frac{1}{\mu_j - \sum_{k=1}^{m} x_{kj}\lambda_k}. \tag{1}$$

Thus, the expected response time of user i can be expressed as:

$$\mathcal{R}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} x_{ij}\mathcal{D}_j = \sum_{j=1}^{n} \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i}, \tag{2}$$

where, $\mu_j^i = \mu_j - \sum_{k=1,k\neq i}^{m} x_{kj}\lambda_k$ is the available service rate of server j for player i.

Remark 1. In this model, the objective is to find a feasible load balancing strategy $\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$ such that the expected response time $\mathcal{R}_i$ of each user $i \in \mathcal{M}$ is minimized.

3.3. Cost model

Dispatchers are interested in minimizing the total cost incurred for them: a user wants to distribute their jobs to servers so that the total cost (the amount of money she pays to the service provider for the servers used) can be minimized.

We consider the fixed pricing model motivated by Amazon EC2 on-demand instances, where a user needs to pay to compute capacity by per unit time. In this pricing model, the prices of servers are fixed by the service provider in advance. The cost incurred for the user of a server is the amount of money that the user needs to pay for per unit use of that server. In this paper, we assume that the per unit time utilization cost of a server is directly proportional to the service rate. This implies that the servers with high service rate are priced high while prices of servers with low service rate are low. This follows from the fact that a high-end server always provides better performance than low-end one, but with higher cost.

The price of per-unit service rate of each server is determined according to their relative service rate. The per-unit service rate price of server j is determined as:

$$p_j = \sigma_j \times \xi, \tag{3}$$

where, σ_j is relative service rate of server j, which is defined as the ratio of service rate of server j to the service rate of the slowest server of the system, i.e., $\sigma_j = \frac{\mu_j}{\min_{\forall j \in \mathcal{N}}(\mu_j)}$, and $\xi \in \mathbb{R}^+$ is a constant. The following example illustrates how the per unit cost of each server is decided in our model.

Example 1. Consider a system that consists of three servers with $(\mu_1, \mu_2, \mu_3) = (5, 10, 20)$. The relative service rate of each server is $\{\sigma_1 = 1, \sigma_2 = 2, \sigma_3 = 4\}$. Let the value of ξ be 10. The values of per unit cost of each server is determined using (3) and is calculated as follows:

$$\begin{cases} p_1 = \sigma_1 \times \xi & = 1 \times 10 & = 10 \\ p_2 = \sigma_2 \times \xi & = 2 \times 20 & = 20 \\ p_3 = \sigma_3 \times \xi & = 4 \times 10 & = 40. \end{cases}$$

Given the vector of server prices $\mathcal{P} = (p_1, p_2, \dots, p_n)$ and the joint load balancing strategy profile $\mathcal{X} = (\mathbf{x}_i, \mathbf{x}_{-i})$, the monetary cost incurred by user i at server j is

$$\mathcal{J}_i^j(\mathbf{x}_i, \mathbf{x}_{-i}) = p_j x_{ij} D_j(\mathbf{x}_i, \mathbf{x}_{-i}). \tag{4}$$

The total monetary cost incurred by user i is the sum of the monetary costs incurred for all the servers that are used by it. Thus, the total monetary cost incurred by user i is expressed as:

$$\mathcal{J}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} \mathcal{J}_i^j(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} \frac{p_j x_{ij}}{\mu_j - x_{ij}\lambda_i}. \tag{5}$$

Remark 2. Given the price vector $\mathcal{P} = (p_1, p_2, \dots, p_n)$, the objective is to find a feasible load balancing strategy $\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$ such that the total cost $\mathcal{J}_i$ of each user $i \in \mathcal{M}$ is minimized.

3.4. Problem formulation

We aim to address a load balancing problem with two conflicting objectives: (i) minimizing the expected response time of each dispatcher and (ii) minimizing the total monetary cost incurred by each dispatcher. The reason behind the conflict is as follows. For the ease of understanding the conflict between the objectives, we provide an illustrative example in the next subsection.

3.4.1. Motivating example

Given a distributed system that consists of three dispatchers and three servers with $(\lambda_1, \lambda_2, \lambda_3) = (4, 10, 14)$ and $(\mu_1, \mu_2, \mu_3) = (5, 10, 20)$. The per unit service rate price of servers is $(p_1, p_2, p_3) = (25, 50, 100)$. Three joint strategy profiles, referred as $\mathcal{X}^1$, $\mathcal{X}^2$, and $\mathcal{X}^3$, are listed below.

$$\mathcal{X}^1 = \begin{matrix} x_{11}^1 = 0.04, & x_{12}^1 = 0.05, & x_{13}^1 = 0.91 \\ x_{21}^1 = 0.13, & x_{22}^1 = 0.28, & x_{23}^1 = 0.59 \\ x_{31}^1 = 0.14, & x_{32}^1 = 0.28, & x_{33}^1 = 0.58 \end{matrix}$$

$$\mathcal{X}^2 = \begin{matrix} x_{11}^2 = 0.21, & x_{12}^2 = 0.30, & x_{13}^2 = 0.49 \\ x_{21}^2 = 0.17, & x_{22}^2 = 0.28, & x_{23}^2 = 0.54 \\ x_{31}^2 = 0.14, & x_{32}^2 = 0.29, & x_{33}^2 = 0.57 \end{matrix}$$

$$\mathcal{X}^3 = \begin{matrix} x_{11}^3 = 0.09, & x_{12}^3 = 0.27, & x_{13}^3 = 0.63 \\ x_{21}^3 = 0.13, & x_{22}^3 = 0.28, & x_{23}^3 = 0.59 \\ x_{31}^3 = 0.14, & x_{32}^3 = 0.28, & x_{33}^3 = 0.58 \end{matrix}$$

For each joint strategy profile, we consider three different criteria that include the expected response time of a dispatcher, expected monetary cost incurred by a dispatcher, and the utilization of a server. Expected response time and monetary cost are determined using Eqs. (2) and (5). Server utilization refers to the average percentage of time that a server is busy. The utilization of server j, ρ_j, is defined as the ratio of load assigned to server j to its processing rate (μ_j), i.e.,

$$\rho_j = \frac{\sum_{i=1}^{m} x_{ij}\lambda_i}{\mu_j}. \tag{6}$$

The values of expected response time, expected costs and server utilization for three different strategy profiles are given in Eqs. (7), (8), and (9), respectively.

$$\begin{cases} \mathcal{R}_1^1 = 0.43, & \mathcal{J}_1^1 = 161.33, & \rho_1^1 = 68.40\% \\ \mathcal{R}_2^1 = 0.43, & \mathcal{J}_2^1 = 127.26, & \rho_2^1 = 69.20\% \\ \mathcal{R}_3^1 = 0.43, & \mathcal{J}_3^1 = 126.19, & \rho_3^1 = 88.30\%. \end{cases} \tag{7}$$

$$\begin{cases} \mathcal{R}_1^2 = 0.73, & \mathcal{J}_1^2 = 120.47, & \rho_1^2 = 91.11\% \\ \mathcal{R}_2^2 = 0.66, & \mathcal{J}_2^2 = 115.42, & \rho_2^2 = 80.57\% \\ \mathcal{R}_3^2 = 0.59, & \mathcal{J}_3^2 = 111.12, & \rho_3^2 = 76.85\% \end{cases} \tag{8}$$

$$\begin{cases} \mathcal{R}_1^3 = 0.37, & \mathcal{J}_1^3 = 104.39, & \rho_1^3 = 72.97\% \\ \mathcal{R}_2^3 = 0.39, & \mathcal{J}_2^3 = 102.71, & \rho_2^3 = 78.92\% \\ \mathcal{R}_3^3 = 0.40, & \mathcal{J}_3^3 = 102.40, & \rho_3^3 = 82.30\% \end{cases} \tag{9}$$

Here, for joint strategy profile k, $\mathcal{R}_i^k/\mathcal{J}_i^k$ denotes the expected response time/ expected monetary cost of dispatcher i, and ρ_j^k denotes the utilization of server j. Some observations on the results shown in Eqs. (7)–(9) are listed below.

- Form the results given in Eqs. (7), (8), and (9), it is clear that joint strategy profile 3 is the best in terms of response time and monetary cost for all three dispatchers. Moreover, joint strategy profile 1 is better than strategy profile 2 in terms of response time for all three dispatchers, while strategy profile 2 is better than 3 in terms of monetary cost for all the dispatchers.
- From the results of Strategy profile 1 and 3 (given in Eqs. (7) and (9)), it can be observed that dispatcher 1, in the pursuit of minimum response time, targets the faster server (i.e., server 3) and allocates 91% of its workload to server 3 only. As a result, the response time of dispatcher 1 gets increased by $\approx$ 16% (from 0.37 to 0.43) in comparison to strategy profile 3, and the monetary cost incurred by it also increased by $\approx$ 55% (from 104.39 to 161.33) in comparison to strategy profile 3. The underlying reason behind this poor performance is the fact that the fastest server (which is also having the highest per-unit service rate price) is excessively utilized.
- From the results of Strategy profile 2 (given in Eq. (8)), it can be observed that in the pursuit of minimum monetary cost, if a dispatcher targets a cheaper server (which is also a slower one), then the minimum cost is achieved, but at the cost of increasing the response time. For instance, in strategy profile 2, dispatcher 1 allocates 21% of its workload to server 1, 30% to server 2 and 49% to server 3. As a result, the value of monetary cost incurred by dispatcher 1 gets decreased by $\approx$ 25% (from 161.33 to 120.47), but at the same time, its response time gets increased by $\approx$ 95% (from 0.43 to 0.73).

To summarize the results discussed above, it can be concluded here that if we target one objective and overlook the other, the targeted objective can be improved but at the cost of deteriorating the overlooked objective. Thus, in order to optimize both the objectives simultaneously, we need to achieve a better trade-off between both the objectives.

3.4.2. Problem statement

Our goal is to achieve a balanced trade-off between these two objectives where the values of both the objectives are optimized. Therefore, to find a feasible strategy, we need to consider both the objectives in an integrated manner by applying the weighted-sum method —we multiply each objective by a weight factor and sum up both the objectives.

It is noteworthy to mention that both the objective functions (given in (2) and (5)) are represented by different measure unit (e.g., $\mathcal{R}_i$ is measured in sec. and $\mathcal{J}_i$ is measured in \$ (USD)). So, it is not possible to combine them directly. To bring both the objectives in the same scale, we normalize the price values and make them unit free. The normalization of the price value p_j of a server is formulated as:

$$c_j = \frac{p_j - \min_{\forall j \in \mathcal{N}}(p_j)}{\max_{\forall j \in \mathcal{N}}(p_j) - \min_{\forall j \in \mathcal{N}}(p_j)} + \delta \tag{10}$$

Here δ is small positive value to make all values strictly positive.

In this way, both the objective can be measured by the same measurement unit, i.e., seconds; and conceptually we can combine them. The weighted objective function for the bi-objective load balancing problem can thus be formulated as follows.

The first objective is

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, \mathcal{R}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i}, \tag{11}$$

and the second objective is

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, \mathcal{J}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} c_j \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i}. \tag{12}$$

After combining both the objectives, we get

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, f_i(\mathbf{x}_i, \mathbf{x}_{-i}) = w_1\mathcal{R}_i(\mathbf{x}_i, \mathbf{x}_{-i}) + w_2\mathcal{J}_i(\mathbf{x}_i, \mathbf{x}_{-i}), \tag{13}$$

where, weight factors $0 \le w_1, w_2 \le 1$ such that $w_1 + w_2 = 1$, denotes the relative importance of each objective.

Thus, with the given modeling and distribution of the job among the servers, we shall now formally state the problem we are attempting to solve. *Given the number of servers n, the number of dispatchers m, the service rates of the servers $\mu_1, \mu_2, \ldots, \mu_n$, the prices of the servers $p_1, p_2, \ldots, p_n$, the jobs generation rates of the dispatchers $\lambda_1, \lambda_2, \ldots, \lambda_m$, and the weights preferences w_1, w_2, find a feasible joint strategy profile $\mathcal{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ such that the objective function value, $f_i(\mathcal{X})$, of each dispatcher is minimized.*

4. Game formulation and analysis

In this section, we formulate the load balancing problem as a non-cooperative game between the users. We characterize the best response strategy of each player and propose an algorithm to compute this strategy. Then, we propose a decentralized algorithm to solve the game.

4.1. Game formulation

A non-cooperative game is defined by a set of rules that guides the behavior of each player and helps her in selecting a strategy according to her preference over a set of possible strategies. A player is free to take any decision to progress in the game. The utility of the player is determined not only by its decision but also affected by the decision of other players participating in the game.

From (13), it can be observed that the load assignment decision of a user is influenced by the strategies of other users. Each user has its level of satisfaction and is self-interested in the sense that it is only interested in minimizing its objective function value. In fact, in this load distribution scenario, users are competing with each other to assign their load to the servers in such a way that their objectives can be optimized. From a game-theoretic point of view, such a competition is considered as an instance of a non-cooperative game [4]. After the competition, users arrive at a point where none of them wants to change their decisions. This stable point is known as the Nash Equilibrium (NE) in game theory.

We now define the non-cooperative load balancing game as follows.

Definition 2 (*Non-cooperative Load Balancing Game*)**.** A non-cooperative load balancing game can be formally defined as the tuple $\mathcal{G} = \langle \mathcal{M}, (\mathbf{x}_i)_{i\in\mathcal{M}}, (f_i)_{i\in\mathcal{M}} \rangle$, where $\mathcal{M} = \{1, \ldots, m\}$ is a set of players, $\mathbf{x}_i$ is the strategy profile of player $i \in \mathcal{M}$, and the penalty function of each player i is f_i.

In this paper, users are regarded as players, The penalty function of each player i ($i \in \mathcal{M}$) is f_i (given in (13)). The strategy profile of each player i is $\mathbf{x}_i = (x_{11}, \ldots, x_{1n})$.

Remark 3. In general, the objective function of players in game-theoretic settings is referred to as the utility function and players goal is to maximize the utility function. In our formulation, the load balancing problem is formulated as the problem of minimization. Therefore, in this paper, rather than referring it as a utility function, we shall refer to it as a penalty function.

In this game, the aim of a player i, given the other players' strategies denoted as $\mathbf{x}_{-i}$, is to select a strategy $\mathbf{x}_i$ so that its penalty $f_i(\mathbf{x}_i, \mathbf{x}_{-i})$ is minimized. The solution concept of this game is Nash equilibrium, which is defined below.

Definition 3 (*Nash Equilibrium*)**.** A joint strategy profile $\mathcal{X}^* = (\mathbf{x}_i^*, \mathbf{x}_{-i}^*)$ is called Nash equilibrium of the above defined load balancing game $\mathcal{G}$, if for all $i \in \mathcal{M}$:

$$\mathbf{x}_i^* \in \underset{\mathbf{x}_i}{\operatorname{argmin}}\, f_i(\mathbf{x}_i, \mathbf{x}_{-i})$$

At Nash equilibrium, no player can further minimize its penalty by choosing a different strategy while keeping the strategies of others fixed.

The equilibrium strategy profile $\mathcal{X}^*$ (Nash equilibrium) is found when each user's strategy is the best response strategy to other players' strategies. Thus, the best response strategy of player i is defined as the strategy which provides a minimum penalty for player i, considering that other players' strategies are given. A detailed description for finding the best response strategy for a player is given in the next subsection.

4.2. Best response strategy computation

The problem of computing the optimal strategy for the player i reduces to the problem of computing optimal strategy for a system with one player and n servers (because for player i the strategies of others are kept fixed). So, for each player $i \in \mathcal{M}$, we solve the following non-linear optimization problem (labeled OPT$_i$)

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, f_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} (w_1 + w_2 c_j) \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i}, \tag{14}$$

$$\text{subject to } \sum_{i=1}^{n} x_{ij} = 1, \tag{15}$$

$$x_{ij} \geq 0, \qquad j = 1, \ldots, n \tag{16}$$

$$x_{ij}\lambda_i < \mu_j^i, \qquad j = 1, \ldots, n. \tag{17}$$

Remark 4. In order to find the solution to OPT$_i$, the strategies of others are kept fixed. There is only one variable in OPT$_i$ that is $\mathbf{x}_i$ (the strategy of player i). So, the solution of OPT$_i$ is the best response strategy of player i.

Remark 5. In objective function f_i (see Eq. (14)), higher value of w_1 implies that the more attention is paid to the user-end (i.e, expected response time of dispatcher), and higher value of w_2 indicates that the more attention is given to fair utilization of servers. Thus, the selection of appropriate values for w_1 and w_2 depends on the objectives that are determined while designing the distributed system. □

4.2.1. Characterization of best response strategy

We first state some important properties of the problem OPT$_i$, and then we characterize the best response strategy of each player $i \in \mathcal{M}$.

Lemma 1. *The set of feasible solutions defined by constraints (15)–(17) is a convex set.*

Proof. It is easy to see that all the constraints are linear. Therefore, the solution set of linear equality (16) and linear inequalities (15) and (17) describes a *polyhedron* [31]. This polyhedron is defined as:

$$\mathbb{S} = \begin{cases} x_{ij} : x_{ij} \geq 0, \quad j = 1, 2, \ldots, n; \\ \quad \sum_{j=1}^{n} x_{ij} = 1; \\ \quad x_{ij}\lambda_i < \mu_j^i, \quad j = 1, 2, \ldots, n, \end{cases} \tag{18}$$

is also known as *convex set*. □

Lemma 2. *The objective function (14) is a strictly convex function.*

Proof. From (14), it can be noted immediately that the objective function is additively separable in the sense that this can be expressed as a sum of terms, each of which depends only on the assignment probability x_{ij}. Further, from (14) we get that

$$\frac{\partial f_i(\mathbf{x}_i, \mathbf{x}_{-i})}{\partial x_{ij}} = \left[\frac{(w_1 + w_2 c_j)}{\mu_j^i - x_{ij}\lambda_i} + \frac{(w_1 + w_2 c_j) x_{ij}\lambda_i}{(\mu_j^i - x_{ij}\lambda_i)^2} \right]$$

and the second derivative is

$$\frac{\partial^2 f_i(\mathbf{x}_i, \mathbf{x}_{-i})}{\partial x_{ij}^2} =$$

$$\frac{(w_1 + w_2 c_j) 2\lambda_i}{\mu_j^i - x_{ij}\lambda_i} \left[\frac{1}{\mu_j^i - x_{ij}\lambda_i} + \frac{x_{ij}\lambda_i}{(\mu_j^i - x_{ij}\lambda_i)^2} \right]$$

Since $c_j > 0$ and $\mu_j^i > x_{ij}\lambda_i$, it is easy to see that $(\partial f_i / \partial x_{ij}) > 0$ and $\partial^2 f_i / \partial x_{ij}^2 > 0$, for all $j = 1, \ldots, n$. In other words, the Hessian matrix of $f_i(\mathbf{x}_i, \mathbf{x}_{-i})$ is positive definite at x_{ij}, for all $j = 1, \ldots, n$. Thus, it can be concluded that objective function (14) is strictly convex [32]. □

From Lemmas 1, and 2, we obtain the following important corollary.

Corollary 1. *The optimization problem OPT$_i$ has a unique minimum solution.*

Proof. By virtue of Lemmas 1, and 2, we may define OPT$_i$ as: $f_i : \mathbb{S} \to \mathbb{R}^+$, i.e., f_i is a strictly convex function defined over convex set $\mathbb{S}$. By applying Soriano theorem [33], it can be shown that there exist a unique minimum point for function f_i. □

In the following theorem, we characterize the best response strategy of player i.

Theorem 1. *Considering that the servers are ordered in non-decreasing order of $\frac{w_1+w_2c_1}{\mu_1^i} \leq \frac{w_1+w_2c_1}{\mu_2^i} \leq \cdots \leq \frac{w_1+w_2c_1}{\mu_n^i}$, the best response strategy of player i (i.e., the solution of OPT$_i$) is given by*

$$x_{ij} = \begin{cases} \frac{1}{\lambda_i}\left(\mu_j^i - \theta\left(\sum_{j=1}^{l} \mu_j^i - \lambda_i \right) \right), & \text{if } j \in [1, l] \\ 0 & \text{if } j \in [l+1, n] \end{cases} \tag{19}$$

where $\theta = \frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{l}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}$ *and l is the server index that satisfies:*

$$l = \sup\left\{k \le n\colon \mu_j^i < \left(\frac{\sum_{j=1}^{k}\mu_j^i - \lambda_i}{\sum_{j=1}^{k}\sqrt{\mu_j^i}}\right)^2\right\} \tag{20}$$

Proof. To improve the readability of the paper, the formal proof of the theorem is relegated to the Appendix Appendix A.1. □

4.2.2. Best response computation algorithm

The procedure of computing the best response strategy of player i, based on Theorem 1, is formally presented in Algorithm 1.

Algorithm 1: Best response algorithm for player i

```
Input  : Available service rate at each server for player $i$ :
         $\mu_1^i, \mu_2^i, \cdots, \mu_n^i$,
         Jobs generation rate of player $i$ : $\lambda_i$
         Normalized price vector : $\mathcal{C} = (c_1, c_2, \cdots, c_n)$
Output:  Best response strategy : $\mathbf{x}_i^*$ = $(x_{i1}, \cdots, x_{in})$
(1)  begin
(2)      Sort the servers in non-decreasing order of
         $\frac{w_1+w_2c_1}{\mu_1^i} \le \frac{w_1+w_2c_2}{\mu_2^i} \le \cdots \le \frac{w_1+w_2c_n}{\mu_n^i}$
(3)      $k \leftarrow n$
(4)      while $\sqrt{\frac{w_1+w_2c_k}{\mu_k^i}} \ge \frac{\sum_{j=1}^{k}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{k}\mu_j^i-\lambda_i}$ do
(5)          $x_{ik} \leftarrow 0$
(6)          $k \leftarrow k-1$
(7)      end
(8)      $\Gamma \leftarrow \frac{\sum_{j=1}^{k}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{k}\mu_j^i-\lambda_i}$
(9)      for $j$ = 1 to $k$ do
(10)         $x_{ij} \leftarrow \frac{1}{\lambda_i}\left(\mu_j^i - \frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\Gamma}\right)$
(11)     end
(12)     return $\mathbf{x}_i$ = $(x_{i1}, \cdots, x_{in})$          ▷ Best response
(13) end
```

The correctness of Algorithm 1 is proved by the following theorem.

Theorem 2. *The load allocation strategy of player i, achieved by Algorithm 1 is an optimal load balancing strategy given by the best-response* $\mathbf{x}_i^* = (x_{i1}, \ldots, x_{in})$.

Proof. To improve the readability of the paper, the formal proof of the theorem is relegated to the Appendix Appendix A.2. □

Lemma 3. *The computational complexity of Algorithm 1 is* $\mathcal{O}(n \log n)$.

Proof. The computational complexity of the basic operations involved in Algorithm 1 is as follows: sorting procedure in step 2 requires $\mathcal{O}(n\log n)$. Recalculating k inside while loop requires $\mathcal{O}(n)$. Assigning the values to x_{ij} inside for loop needs $\mathcal{O}(n)$. Hence, the overall computational complexity of Algorithm 1 is $\mathcal{O}(n\log n)$. □

4.3. Nash equilibrium computation

From an algorithmic point of view, Nash equilibrium is a potential convergence point of the dynamic adjustment process in which players adjust their behavior to that of other players in the game, constantly searching for strategies that will yield them the best response. To compute the Nash equilibrium of load balancing game (Definition 2), some coordination between players is required to acquire the load information from each server [4]. In the next subsection, we suggest a decentralized load balancing algorithm.

4.3.1. CALBA: Cost-Aware Load Balancing Algorithm

In this algorithm, each player refines its strategy by computing the best-response against the given strategies of other players. The players sequentially update their strategies as shown in Fig. 2. The pseudo-code of the proposed algorithm (CALBA) is presented in Algorithm 2

In addition to the notations given in Section 3, we define some more notations which are given below.

- t: iteration index
- $f_i^{(t)}$: player i's penalty at iteration t
- ε: acceptance tolerance
- $\Delta = \left\|\mathbf{F}^{(t)} - \mathbf{F}^{(t-1)}\right\|$: ℓ_1 norm at iteration t, where $\mathbf{F}^{(t)}$ is penalty vector of all the players at iteration t, such that $\mathbf{F}^{(t)} = \left(f_i^{(t)}\right)_{i=1}^{m}$.
- **SEND**$(i, [p, q, r])$: send message $[p, q, r]$ to player i
- **RECEIVE**$(i, [p, q, r])$: receive message $[p, q, r]$ from player i

Algorithm 2: Cost-Aware Load Balancing Algorithm (*CALBA*) for finding Nash equilibrium

```
Input  : service rate of servers: $\mu_1, \mu_2, \cdots, \mu_n$
         Jobs generation rate of usera: $\lambda_1, \lambda_2, \cdots, \lambda_m$
Output:  Load balancing strategy $\mathcal{X}^* = (\mathbf{x_1^*}, \mathbf{x_2^*}, \cdots, \mathbf{x_m^*})$
(1)  begin
(2)      $f_i^{(0)} \leftarrow 0$
(3)      $t \leftarrow 0$
(4)      $\Delta \leftarrow 1$
(5)      temp $\leftarrow 0$
(6)      flag $\leftarrow$ CONTINUE
(7)      left $\leftarrow [(i-2) \bmod m] + 1$
(8)      right $\leftarrow [i \bmod m] + 1$
(9)      while true do
(10)         if (i=1) then                                  ▷ player1
(11)             if $t \ne 0$ then
(12)                 RECEIVE(left, [$\Delta$, $t$, flag])
(13)                 if $\Delta < \varepsilon$ then
(14)                     SEND(right, [$\Delta$, $t$, STOP])
(15)                     Exit
(16)                 end
(17)                 temp$\leftarrow$0
(18)                 $t \leftarrow t+1$
(19)             end
(20)         else                                           ▷ other players
(21)             RECEIVE(left, [$\Delta$, $t$, flag])
(22)             if flag=STOP then
(23)                 if $i \ne m$ then
(24)                     SEND(right, [temp, $t$, STOP])
(25)                 end
(26)                 Exit
(27)             end
(28)             for $j \leftarrow 1$ to $n$ do
(29)                 $\mu_j^i \leftarrow \mu_j - \sum_{k=1,k\ne i}^{m} x_{kj}\lambda_i$
(30)             end
(31)             $\mathbf{x}_i^* \leftarrow$ using Alg. 1
(32)             Evaluate $f_i^t \leftarrow$ using (13)
(33)             temp$\leftarrow$ temp+$\left\|f_i^{(t)} - f_i^{(t-1)}\right\|$
(34)             SEND(right, [temp, t, CONTINUE])
(35)         end
(36)     end
(37) end
```

Description of Algorithm 2: To implement Algorithm 2, we assume that all the players are connected in a unidirectional ring

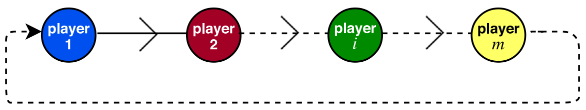


Fig. 2. m players in an iteration of Algorithm 2.

as shown in Fig. 2. The algorithm is initiated by one of the players called the initiating player. The initiating player i computes its best-response strategy (using Algorithm 1) by fixing the other strategies. In order to evaluate its best-response, player i does the following. Player i receives the message from its anti-clockwise neighbor. If the received message is termination message (i.e., the flag is STOP) then player i will pass the termination message to its clockwise neighbor and declare initially assigned strategy as its best response strategy. If the message received by player i from its anti-clockwise neighbor is not a termination message (i.e., the flag is CONTINUE) then player i evaluates the available service rate of each server (step 28–30) by observing their running queue length and computes the best response strategy (step 31), penalty value (step 32), and norm (step 33). Then, player i sends the updated norm value to its clockwise neighbor.

An iteration is completed when the initiating player receives a message from its anti-clockwise neighbor. This best response strategy computation of players goes on for several iterations. After several iterations, the algorithm converges to equilibrium and then terminates.

Lemma 4. *The message complexity of Algorithm 2 is $\mathcal{O}(mn)$.*

Proof. The message complexity of Algorithm 2 for one iteration is as follows. To know the status of the running queues of a server, a player requires 2 messages: one message from player to server and another from server to player. In this way, each player requires a $2n$ number of messages to gather the available processing rate of all servers. Next, to share its updated norm value with the clockwise neighbor, each user requires one message. Thus, in each iteration, all the players require $2mn+m$ messages, where m and n are the numbers of players and servers respectively. Hence, the message complexity of Algorithm 2 is $\mathcal{O}(mn)$. □

Remark 6. According to [4,6,7,9,29], it is not possible to give a theoretical proof for the convergence of such an iterative algorithm. However, Orda et al. [34,35] studied the convergence of the distributed load balancing algorithm in the context of routing problems in parallel networks. In [34,35], the authors have investigated the convergence for two players and two servers game and explicitly pointed out that "the results obtained for two players game is not readily extendible to more general cases". Hence, the convergence proof of such distributed load balancing algorithms is still an open problem [4]. In Section 5.2.1, we simulate the proposed algorithm (*CALBA*) and the experimental results verify that the convergence of *CALBA* is similar to that of existing algorithms [4,6,7,9,29].

5. Experimental results

In this section, to ascertain the effectiveness of our proposed algorithm, we perform a experimental study and provide numerical results. We assume that algorithm stops (i.e., the convergence has occurred) when the threshold value of acceptance tolerance, i.e., $\varepsilon \leq 10^{-5}$. The value of positive constant for δ (used for normalization formula (10)) is set to 0.01 for each experiment.

5.1. Performance metrics

In order to assess the quality of solutions obtained by our algorithm, four performance metrics: (i) Fairness Index (FI), (ii) Global response time (GRT), (iii) Global Monetary cost (GMC), and (iv) Load Imbalance Percentage (PLI), are used. A brief description of the metrics follows next.

- Fairness Index (FI): it measures the equality of players expected response time [4,36]. FI defined as follows.

$$\text{FI} = \Big(\sum_{i=1}^{m} \mathcal{R}_i\Big)^2 / m \sum_{i=1}^{m} \mathcal{R}_i^2, \tag{21}$$

  where $\mathcal{R}_i$ is the expected response time of a player i (defined in (2)). The value of FI lies between 0 and 1. Ideally, if expected response time of each player is same, then this metric should evaluate to 1.
- Global Response Time (GRT): it measures the expected response time of the system as a whole [10]. It is defined as

$$\text{GRT} = 1/\lambda_T\Big(\sum_{i=1}^{m} \lambda_i \mathcal{R}_i\Big) \tag{22}$$

  λ_T is the aggregate job arrival rate in the system, i.e., $\lambda_T = \sum_{i=1}^{m} \lambda_i$.
- Global Monetary cost (GMC): it is the total cost incurred in the system, i.e., it is the total amount of money that is incurred by the service provider in the system. It is determined as:

$$\text{GMC} = \sum_{j=1}^{n} p_j(1/\mu_j - \sum_{k=1}^{m} x_{kj}\lambda_k) \tag{23}$$

- Percent Load Imbalance (PLI):it measures how evenly the load is distributed to the servers by the players [37]. The percent load imbalance is evaluated as:

$$\text{PLI} = \left(\frac{\rho^{max}}{\widehat{\rho}} - 1\right) \times 100\% \tag{24}$$

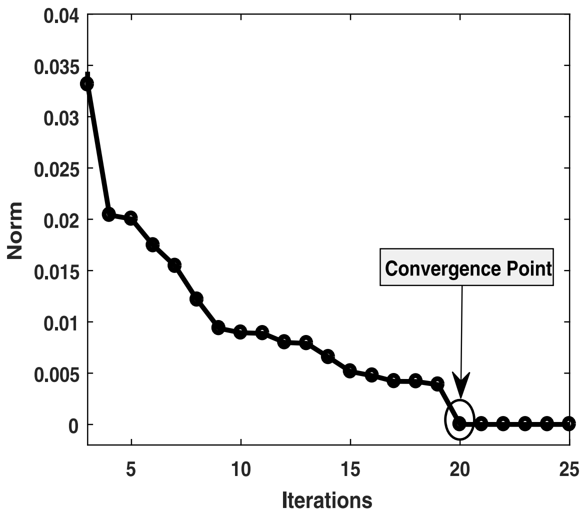
  where $\rho^{max} = \max\{\rho_j : j = 1, \dots, n\}$ denotes the maximum utilization of a server and $\widehat{\rho}$ denotes the average utilization of the servers. Ideally, if each system is equally utilized, this metric should evaluate to zero.

5.2. Performance evaluation

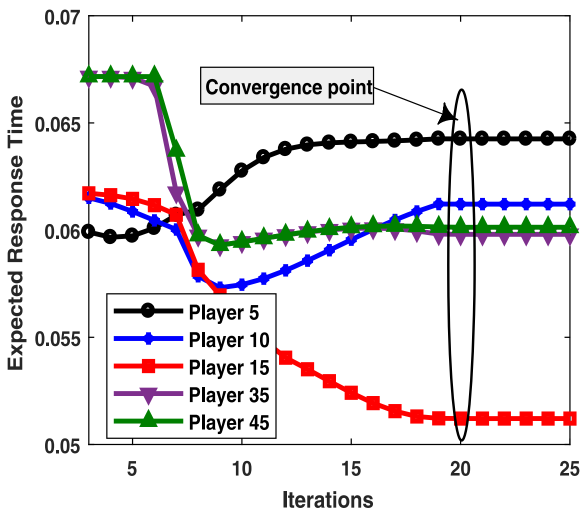
In this section, we perform the experiments and provide their results to manifest the performance of our algorithm.

5.2.1. Convergence analysis

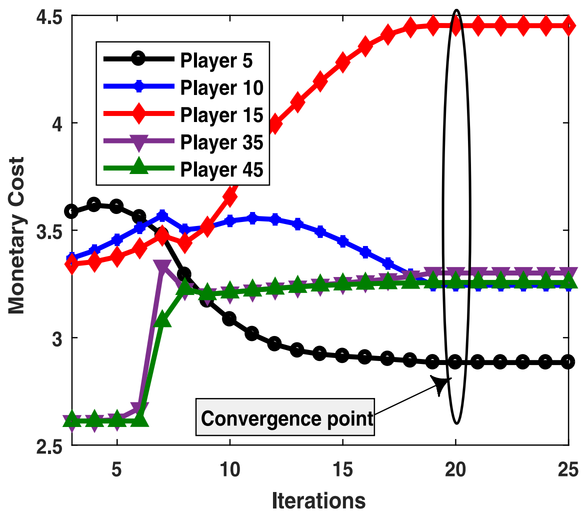
In this section, we demonstrate the convergence behavior of our proposed algorithm. For this, we consider a heterogeneous system of 100 servers and 50 users. The service rate of 100 servers is given in Table 2. The first row contains the indexing of the servers, the second row contains the service rate of corresponding servers, and the third row contains the relative service rate of corresponding servers. The job arrival fraction of players is given in Table 3. For each experiment, the total job arrival rate λ of the system is determined using total service rate of the system (i.e., $\sum_{j=1}^{n} \mu_j$) and system utilization (ρ) as follows $\lambda = \rho \times \sum_{j=1}^{n} \mu_j$. The job arrival rate λ_i of each player is determined as $\lambda_i = \phi_i \times \lambda$. For the experiments, the per unit price for each server is calculated using the formula $p_j = \xi \times \sigma_j$, the system utilization ρ is set to 0.7 (i.e., 60%), the value of ξ is set to 10, the weight $w_1 = w_2 = 0.5$, and the number of iterations are fixed to 30.



(a) convergence of norm



(b) convergence of specific response time



(c) convergence of specific monetary cost

Fig. 3. Convergence process of the CALBA.

Table 2
Server configuration I: service rate μ_j of different servers.

Servers	1–10	11–50	51–70	71–92	93–100
Service rate (μ_j)	10	20	40	50	100
Relative service rate (σ_j)	1	2	4	5	10

Table 3
Job arrival fraction ϕ_i for each player.

Players	1–5	6–10	11–30	31–35	36–50
ϕ_i	0.08	0.05	0.014	0.008	0.002

In Fig. 3, we present three different instances of the convergence process of the proposed algorithm. Fig. 3a shows the pattern that how the value of norm changes throughout the iteration. In Figs. 3b and 3c show the iteration wise trends of users expected response time and the monetary cost (the total amount of money paid by users to the service provider), respectively. Specifically, Figs. 3b and 3c show the iteration wise trends of 5 players (player5, player10, player15, player35, and player45). In terms of jobs generation rates, these selected players lie in five different groups: (i) player5 — 172.8 job/sec., (ii) player10 — 108 job/sec., (iii) player15 — 30.24 job/sec., (iv) player35 — 17.28 job/sec., and (v) player45 — 4.32 job/sec. From Fig. 3a, we can observe that as the number of iteration increases the norm value decreases. After 20th the value of norm remains constant which signifies that the algorithm has reached the convergence.

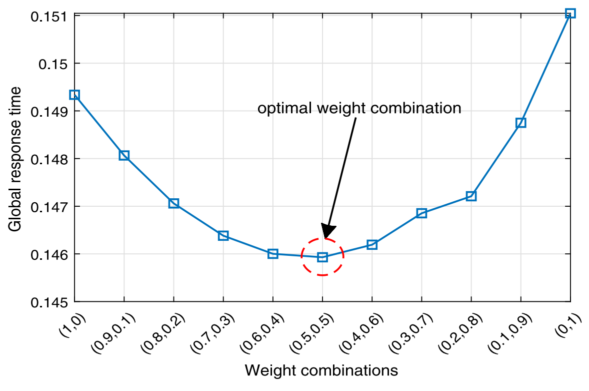
Figs. 3b and 3c show iteration wise pattern for expected response time values and monetary cost incurred values of different players. From Fig. 3b, we can observe that the expected response time of some players (player10, player15, player35, and player45) tend to decrease in early iterations and after some iteration (specifically after 10th) their expected response time increases with the increase of iterations. On the other hand, the expected response time of player5 increases with the increase of the iteration number. From Fig. 3c, we can observe that the monetary cost of a player increases/decreases, if its expected response time decreases/increases. However, all the expected response times and monetary costs reach a stable state after iteration 20. In other words, a player will not succeed in minimizing its penalty by unilaterally deviating its strategy after iteration 20. Hence, the algorithm has reached the Nash equilibrium.

5.2.2. Effect of user weight preferences

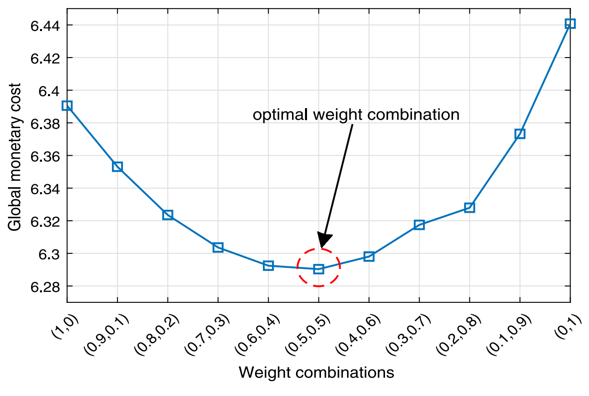
In this section, we analyze the impact of weight factors w_1 and w_2 on the performance of CALBA. We select 11 different combinations of weights (w_1, w_2) as: (1, 0), (0.9,0.1), (0.8, 0.2), (0.7, 0.3), (0.6,0.4), (0.5, 0.5), (0.4, 0.6),(0.3, 0.7), (0.2, 0.8), (0.9, 0.1) and (0, 1). The number of players and the number of servers are 10, and the configuration of servers and players are given in Tables 2 and 3. The system utilization ρ is set to 0.7 (i.e., 60%), the value of ξ is set to 2.

The values of different performance metrics, over different weight combinations, are shown Fig. 4. Figs. 4a and 4b, show weight combination versus global response time (GRT) and global monetary cost (GMC), respectively. In Figs. 4a and 4b that the values of GRT and GMC achieved w.r.t. $(w_1, w_2) = (0.5, 0.5)$ are way much better (less) than that w.r.t. to other combinations. This signifies that giving equal preference to both the objectives is the best way to achieve the best overall system-wide performance.

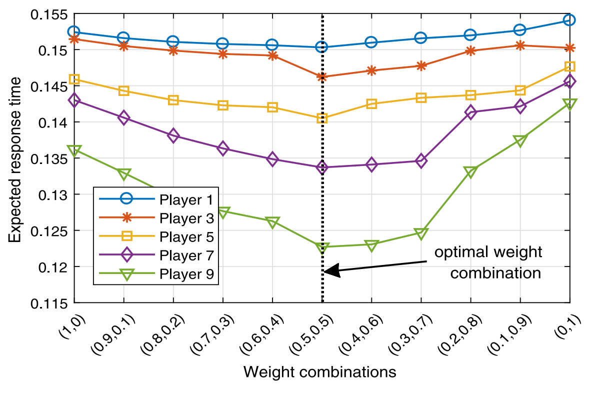
Figs. 4c and 4d, show weight combination versus expected response time (given in Eq. (2)) and expected monetary cost (given in Eq. (5)) for some players, respectively. As illustrated in Fig. 4c, the values of response time for each player decreases with the increment (decrement) of w_2 (w_1). However, after $w_1 = w_2 =$



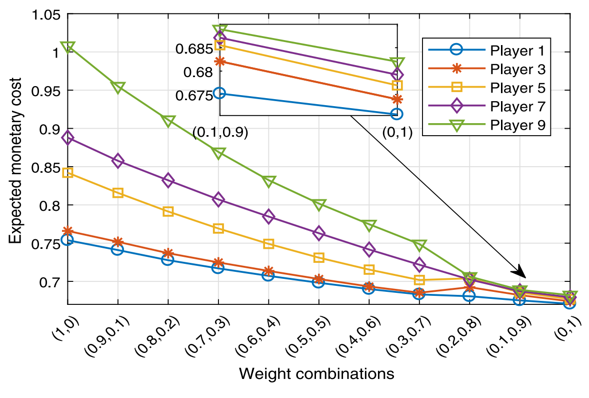
(a) Weight combunation versus global monetary cost



(b) Weight combunation versus global response time



(c) Weight combunation versus expected response time



(d) Weight combunation versus expected monetary cost

Fig. 4. Comparison results of different performance metrics for different weight combinations.

0.5, this trend reverses. In Fig. 4d, the value of cost incurred by each player decreases with the increment (decrement) of w_2 (w_1). It can be observed that minimum response time is achieved at the cost of maximum cost and vice versa. For instance, for player 7, the response time obtained at $w_1 = w_2 = 0.5$ is 7% less than that of (w_1, w_2) =(1,0) and 9% less than that of (w_1, w_2) =(1,0), while the response time obtained at $w_1 = w_2 = 0.5$ is 17% less than that of (w_1, w_2) =(1,0) and 10% more than that of (w_1, w_2) =(1,0). This is because the fact that as the value of w_1 decreases (or value of w_2 increases), the players are more concerned about minimizing their cost, and they focus on cheaper servers (i.e., slower servers) and make them congested. As a result, a player with higher job generation rates experiences high response time than a player with lower job generation rate. In addition to that, when the priorities for both the objectives are same (i.e., $w_1 = w_2 = 0.5$), players distribute their load to costly servers also (high-end servers) rather targeting only cheaper servers. As a result, better system performance is achieved.

Having observed the empirical results discussed above, it is clear that the selection of the proper weight combination has an impact on the performance of the algorithm, but determining the optimal combination of weights is not possible empirically. Finding the optimal weight combination is another optimization problem which is beyond the scope of this paper. Nonetheless, it would be reasonable enough to argue that giving equal priorities to both the objectives is a fair choice. In this way, for all the experiments carried out in this paper, we fix the weights as $w_1 = w_2 = 0.5$.

5.3. Performance comparison

To assess the relative performance of our proposed algorithm, we compare it with three other load balancing schemes. A brief description of these schemes is as follows:

- Non-cooperative load balancing scheme [4] : in this scheme, each player wants to minimize its expected response time without considering the cost factor. In this paper, we refer it as *MinRT*. The strategies $\mathbf{x}_i$ for each player are determined by solving the following problem:

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, \mathcal{R}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i},$$

s.t. the constraints (15), (16), and (17). (25)

- Cost minimization scheme (*MinCost*): in this scheme, players are interested in minimizing their incurred monetary cost in using the servers without caring about the response time. In this scheme, the strategies $\mathbf{x}_i$ for each player are obtained by solving the following problem:

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, \mathcal{J}_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \sum_{j=1}^{n} c_j \frac{x_{ij}}{\mu_j^i - x_{ij}\lambda_i},$$

s.t. the constraints (15), (16), and (17). (26)

- Global price minimization scheme (*GPMS*) [15]: This scheme intended to minimize the overall cost for executing all the jobs in the system. We modified the *GPMS* to suit the heterogeneous system model considered in this paper. The strategies $\mathbf{x}_i$ for each player are determined by solving the following non-linear optimization problem:

$$\underset{\mathbf{x}_i}{\operatorname{argmin}}\, J(\mathbf{x}_i, \mathbf{x}_{-i}) = \frac{1}{\sum_{i=1}^{m} \lambda_i} \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{c_j x_{ij} \lambda_i}{\mu_j^i - x_{ij}\lambda_i}, \quad (27)$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, m \tag{28}$$

$$x_{ij} \geq 0, \quad i = 1, \ldots, m, j = 1, \ldots, n \tag{29}$$

$$\sum_{i=1}^{m} x_{ij}\lambda_i < \mu_j^i, \quad j = 1, \ldots, n. \tag{30}$$

5.3.1. *Effect of system utilization*

In this experiment, the impact of system utilization (ρ) on the load balancing schemes is studied. The system utilization is defined as the ratio of the total job arrival rate to the total processing rate of the system: $\rho = \sum_{i=1}^{m} \lambda_i / \sum_{j=1}^{n} \mu_j$. In this part of experiments, we vary the value of system utilization from 0.5 to 0.9 (i.e., 50% to 90%). We consider a heterogeneous system of 100 servers and 50 users. The service rate of servers and jobs arrival rate of players are given in Tables 2 and 3, respectively. We fix the value ξ to 5. The numerical results obtained by the algorithms are presented in Fig. 5.

Fig. 5a presents the fairness index (FI) for different values of system utilization (from 50% to 90%) based on the load balancing decisions taken by *CALBA*, *MinRT*, *MinCost*, and *GPMS*. This varying utilization corresponds to a total jobs arrival rate on the system ranging from 1800 to 3240 jobs/s. It can be seen from Fig. 5a that *CALBA* has an FI value of almost 1 across the entire utilization range. It is the inherent advantage of *CALBA* that it guarantees the same expected response time to all the players. The FI value for *MinRT* is better than *MinCost* and *GPMS*. The FI for *MinCost* varies from 0.975 (at low utilization,i.e., 50%) to 0.96 (at high utilization, i.e., 90%). *GPMS* yields the worst performance in terms of FI value because its targets only cheaper servers (slower servers).

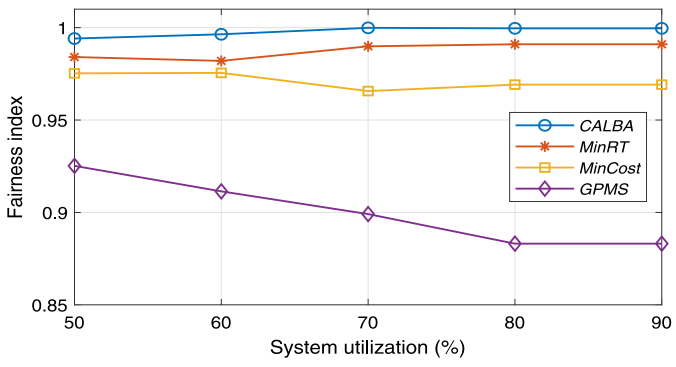
Fig. 5b shows the global response time (GRT) for different values of system utilization based on the load allocation strategies provided by *CALBA*, *MinRT*, *MinCost*, and *GPMS*. The GRT value provided by *CALBA* is the lowest for the entire range of utilization. The GRT value for *MinRT* is higher than *CALBA* and less than *MinCost* and *GPMS*. However, the GRT value obtained by *MinCost* is higher than all the approaches.

Fig. 5c shows the global monetary cost (GMC) for varying system utilization. As can be seen in Fig. 5c, *CALBA* is the best among all in terms of GMC value. Surprisingly, *MinCost* whose objective is to reduce the GMC, is the worst performer in GMC itself. This reason behind is that *MinCost* allocates the bulk of load to less powerful servers and so its GMC is higher than the others.

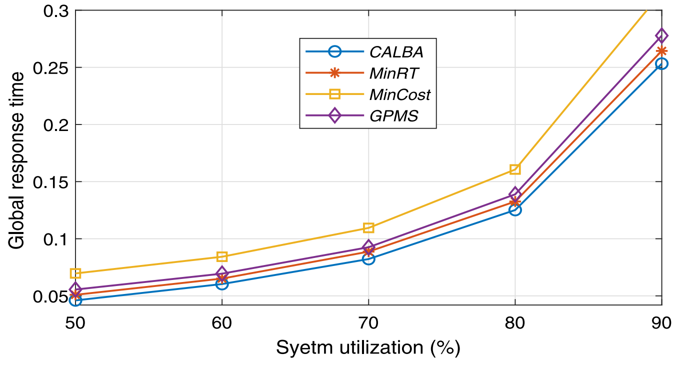
In Fig. 5d percentage load imbalance (PLI) between different servers over various utilization level for each scheme is shown. It can be observed from Fig. 5d that the PLI for *MinCost* is the lowest (best) because due to its inherent tendency of reducing the cost, it does not target only high power servers (costly servers), but also allocate a high amount of load to less powerful servers. The PLI value for *CALBA* is always between *MinCost* and *GPMS*. However, PLI for *CALBA* at 70% is 35%, and depending upon the requirement of decision-maker (system designer), this value of PLI may be close or above the minimum acceptable level. We can, therefore, conclude that proposed *CALBA* is more scalable than *MinCost*, *GPMS*, and *MinRT* in terms of utilization.

5.3.2. *Effect of system heterogeneity*

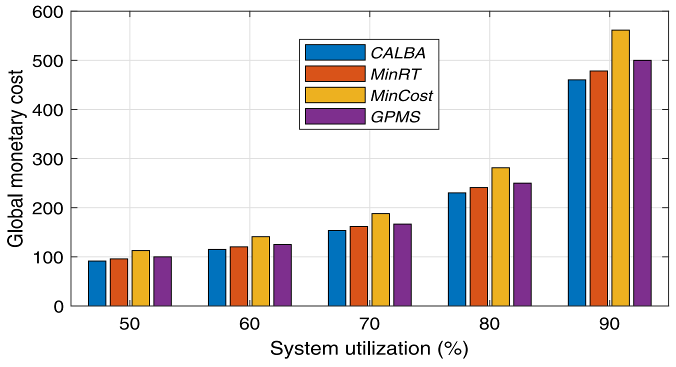
The heterogeneity of a distributed system is characterized by the processing speed of different servers. One of the prevalent measures to quantify the level of heterogeneity is speed skewness which is defined as the ratio of the maximum service rate to the minimum service rate of the servers. Speed skewness is proportional to the level of heterogeneity, i.e., a system with the high value of skewness has a high level of heterogeneity.



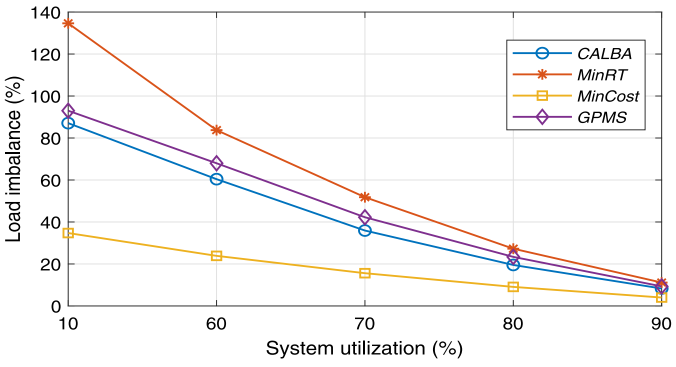
(a) System utilization Versus Fairness index



(b) System utilization Versus Global response time



(c) System utilization Versus Global monetary cost



(d) System utilization Versus Load imbalace

Fig. 5. Effect of system utilization: performance comparison of different schemes over varying system utilization.

Table 4
Server configuration II: service rate μ_j and speed skewness (Skew) of different servers.

Skew	4	16	36	64	100
$\mu_j(j = 1{:}25)$	25	12.5	8.33	6.25	5
$\mu_j(j = 26{:}75)$	50	50	50	50	50
$\mu_j(j = 76{:}100)$	100	200	300	400	500

In this experiment, we observe the effect of heterogeneity on the load balancing schemes for different values of speed skewness. We consider a system of 100 servers whose configuration is given in Table 4. The servers are categorized into three different groups: (i) faster group-represented by servers 1 to 25, (ii) moderate group-represented by servers 26 to 75, and (iii) slower group-represented by servers 76–100. All the other parameters for this experiment is as in the previous experiment.

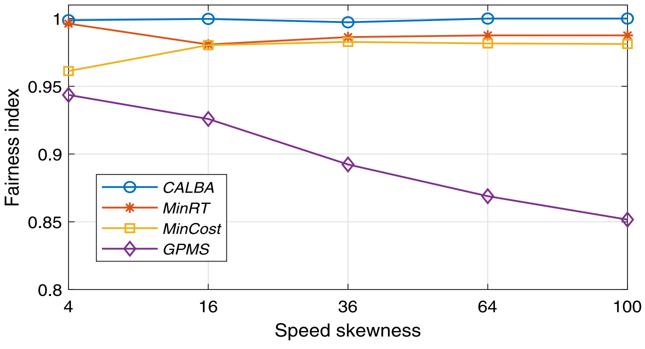
The results obtained by different algorithms are presented in Fig. 6. From Figs. 6a–6c, it can be seen that the performance obtained by *CALBA* is better than the tree competitors: *MinRT*, *MinCost*, and *GPMS*. It can be observed from Fig. 6a that FI for *CALBA* is almost 1 for the entire range of skewness from 4 to 100. The *MinRT* and *MinCost* scheme shows some variations in FI as speed skewness is varied. However, the FI value for *GPMS* decreases with the increase of system heterogeneity.

From Fig. 6b, it can again be observed that GRT for *CALBA* is always the lowest (best) and for *MinCost* is always the highest (worst). The poor performance of *MinRT* is because each player allocates a relatively large amount of workload to powerful servers which cause overloading to them. On the other hand, the performance of *MinCost* is poor because in this case, less powerful servers are overloaded. In Fig. 6c, similar to the previous experiment, GMC for *MinCost* is higher because it overloaded less powerful servers and the increased turn around time results in high cost for the system. In last, from Fig. 6d, it can again be observed that the PLI value for *CALBA* is always between *MinCost* and *GPMS*. However, at skewness = 36, PLI for *CALBA* is about 155% less than *MinRT* and about 14% higher than that of *MinCost*, while at skewness = 100, PLI for *CALBA* is about 185% less than *MinRT* and about 100% less than that of *GPMS*. We can, therefore argue that by using *CALBA* in case of a high level of skewness a better system-wide performance can be achieved.

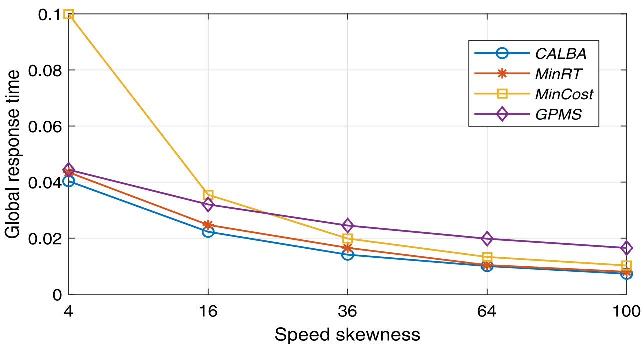
5.3.3. Effect of system size

In this set of experiments, we investigate the impact of system size (number of servers) on the performance of different load balancing schemes. We consider different types of systems which configuration is given in Table 5. For all the systems, the number of users (m) is 100 whose configuration is given in Table 3, and all other parameters are same as in the previous experiment (given in Section 5.3.2). The interpretation of configuration Table 5 is as follows. Each row of the table from 3 to 9 represents a system. For instance, (a) the first occurrence of 3rd row, i.e., 10, denotes the number of servers, (b) second occurrence of 3rd row, i.e., 330, is the total jobs arrival rate in the first system, (c) the third occurrence of 3rd row, i.e., 5, signifies that the number of servers with processing rate 10 is 5. In this set of experiments, the number of servers varies from 10 to 1120. The results obtained by all schemes are plotted in Fig. 7.

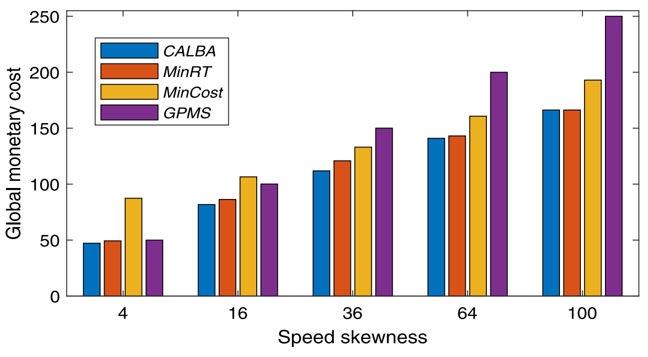
From Figs. 7a–7c, it can be seen that the performance obtained by *CALBA* is better than all the tree contenders. It can be observed from Fig. 6a that FI for *CALBA* is almost 1 for the entire range of system size. The *MinCost* and *GPMS* schemes show some variations in FI as speed skewness varies, however, after certain size (number of servers = 70) it remains constant (FI = 0.8087 for *MinCost* and FI = 0.7602 for *GPMS*). In Fig. 7b, it can be seen that GRT for all the schemes is close for smaller system size (number of servers = 10). Initially, as the system size



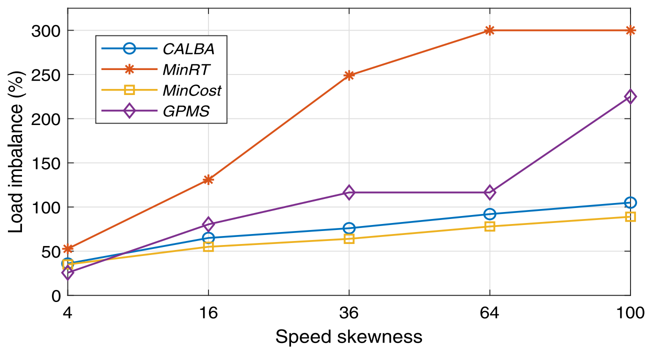
(a) Speed skewness Versus Fairness index



(b) Speed skewness Versus Global response time

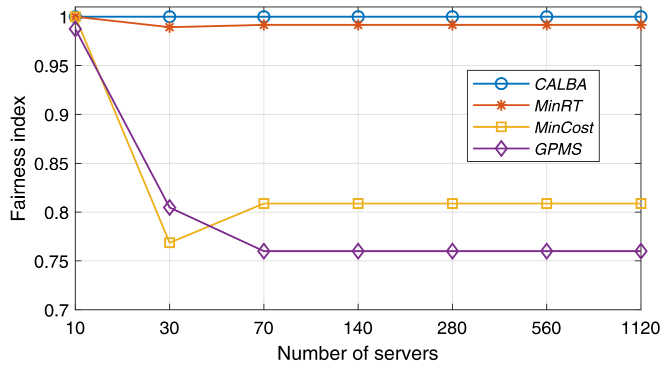


(c) Speed skewness Versus Global monetary cost

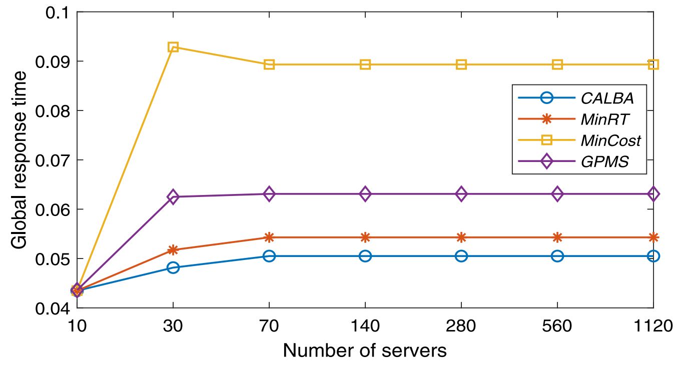


(d) Speed skewness Versus percentage load imbalace

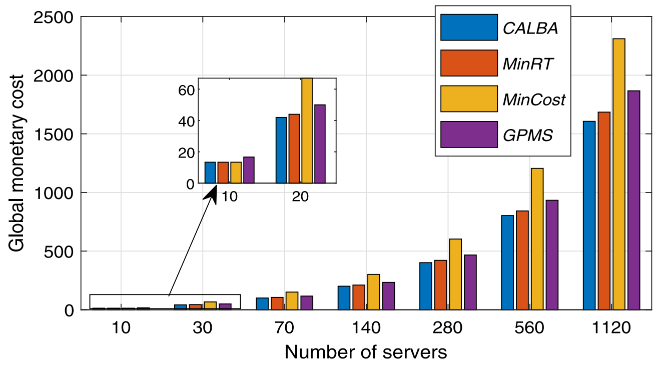
Fig. 6. Effect of system heterogeneity: performance comparison of different schemes over varying speed skewness.



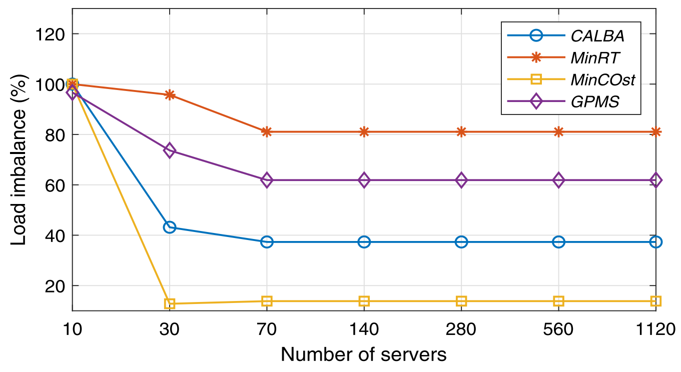
(a) Number of servers Versus Fairness index



(b) Number of servers Versus Global response time



(c) Number of servers Versus Global monetary cost



(d) Number of servers Versus percentage load imbalance

Fig. 7. Effect of system size: Effect of system heterogeneity: performance comparison of different schemes over varying number of servers.

Table 5
System configuration I.

Number of servers (n)	Total jobs arrival rate (λ_T)	Service rate (μ_j)					
		10	20	40	60	80	100
10	330	5	0	0	0	0	5
30	960	10	5	0	0	5	10
70	2220	20	10	5	5	10	20
140	4440	40	20	10	10	20	40
280	8880	80	40	20	20	40	80
560	177 600	160	80	40	40	80	160
1120	35 520	320	160	80	80	1600	320

increases, the GRT for all schemes increases, but after a certain value of system size, it remains constant. A similar trend can also be seen in Fig. 7c. From Fig. 7d, it can be observed that for a system with 10 number of servers the PLI for each scheme is same, i.e, 100%, and further it decreases with the increment of a number of servers in the system. After system 3 (number of servers = 70), PLI for each becomes constant. The reason behind is the system configuration considered, e.g., number of servers with different processing power in the system is two times more than the number of servers in system 3. Hence, in sum, it can be inferred that *CALBA* provides better scalability in terms of system size while maintaining a trade-off between both the cost and response time values.

5.4. Discussion on results

In this section, we further present the following summary remarks concerning with the performance of the proposed *CALBA* approach.

In this paper, to deal with the load balancing problem that arises in a distributed computing system, we proposed a decentralized algorithm called *CALBA*. To test the efficacy of the algorithm, we compared it with three existing approaches from three different aspects, including system utilization, system heterogeneity, and system size. We considered four different performance metrics to gauge the quality of solutions obtained by the algorithms. The summary of the results obtained by the algorithm in terms of each performance metrics is as follows.

1. *Performance analysis in terms of fairness:* Fairness is an important characteristics in modern distributed systems such as Amazon Elastic Compute Cloud and Sun Grid Compute Utility where uses pay for the compute capacity used. In every aspect of the experiments, the value of fairness index obtained by *CALBA* is almost one which has not been achieved by any other competitive algorithm. This clearly indicates that *CALBA* guarantees almost equal response time for all the users independent of the allocated servers.
2. *Performance analysis in global response time:* Global response time refers the overall response time of the system as whole. The value of GRT obtained by *CALBA* is better than every competitive algorithm in each aspect of experiments. Specially, in case of medium system utilization, i.e., 50%, the GRT of *CALBA* is ≈ 20% less than *MinRT*, ≈ 42% less than *MinCost*, and ≈ 27% less than *GPMS*. On the other hand, incase of high system utilization, i.e., 50%, the GRT of *CALBA* is ≈ 5% less than *MinRT*, ≈ 20% less than *MinCost*, and ≈ 12% less than *GPMS*.
3. *Performance analysis in terms of global monetary cost:* Global monetary cost refers the overall monetary cost of the system as whole. In terms of GCT, *CALBA* performs better or commensurable to the other three competitive algorithms in each aspect of experiments. Moreover, in critical

Table 6
Performance comparison of *CALBA* over different performance metrics with three other schemes.

Metrics	*CALBA*	*MinRT*	*MinCost*	*GPMS*
FI	**0.9998 (1)**	0.9898 (2)	0.9656 (3)	0.8991 (4)
GRT	**0.1251 (1)**	0.1325 (2)	0.1606 (4)	0.1388 (3)
GMC	**153.62 (1)**	161.78 (2)	187.96 (4)	166.67 (3)
PLI	**35.92 (2)**	51.79 (4)	**15.60 (1)**	42.25 (3)
Avg. rank	**1.25**	2.5	3	3.25
Overall rank	**1**	2	3	4

system conditions such as high system load, high system heterogeneity, and larger system size, the performance of *CALBA* is much better than the others. This signifies that *CALBA* is maintains the better trade-off between fairness and efficiency.

4. *Performance analysis in terms of percentage load imbalance:* Percentage load imbalance refers how the load is distributed to different servers. In terms of PLI value, *CALBA* performs better than *MinRT* and *GPMS* in each aspect of experiments. However, in critical system conditions such as in case of high system load, i.e., 90% PLI for *CALBA* is $\approx$ 4% higher than *MinCost*, in case of higher system heterogeneity, i.e., 100, PLI for *CALBA* is $\approx$ 15% higher than *MinCost*, and in case of larger system size, i.e., number of servers is 1120, PLI for *CALBA* is $\approx$ 21% higher than *MinCost*. We can, therefore, conclude that depending upon the requirement of decision-maker (system designer), these value of PLI may be close or above the minimum acceptable level.
5. *The overall performance comparison:* to evaluate the overall performance, we compare our *CALBA* with the three other schemes and manifest the performance of our algorithm. In this set of experiments, we consider a heterogeneous system of 100 servers and 50 users. The service rate of servers and jobs arrival rate of players are given in Tables 2 and 3, respectively. We fix the value $\xi = 5$. The values of different metrics obtained by approaches are presented in Table 6. The best result over each performance index is shown in boldface. Further, a rank is assigned to each approach as per the obtained value of performance indices. To compute the ranking of the approaches, first, the ranks of the approaches overall indices is averaged, and then the final ranking is obtained by ordering the average ranks and assigning ranks to the approaches, accordingly. In the 5th and 6th rows of Table 6, average ranking and the final ranking of the schemes are presented.
From Table 6, it can be observed that *CALBA* is ranked 1 for three metrics and rank 2 for one metrics, and therefore the overall ranking of *CALBA* is best (rank 1) among all schemes. The reason behind such performance of *CALBA* lies in the fact that apart from minimizing the response time of players, *CALBA* also minimizes the load imbalance between servers. In other words, unlike *MinRT*, *MinCost* and *GPMS*, *CALBA* neither underutilizes a slower server nor overburdens a faster one. From Table 6, it can be deduced that unlike others, *CALBA* produces an allocation of load which guarantees the equal response time of each player with minimum cost.

6. Conclusion

In this paper, we addressed a load balancing problem in distributed systems that has two conflicting objectives: (i) minimizing the users' expected response time and (ii) minimizing the total monetary cost incurred by each user. We modeled the problem as a non-cooperative game. To find the solution of this game, we first characterize the best response strategy for each player in Theorem 1. Then, we derive Algorithm 1 to compute the best response strategy of each player, and establish the correctness of Algorithm 1 in Theorem 2. To compute the solution of the game (Nash equilibrium), we proposed Algorithm 2: **C**ost-**A**ware **L**oad **B**alancing **A**lgorithm (*CALBA*). The convergence of *CALBA* is analyzed experimentally and we find that it converges to Nash equilibrium after some iterations. Further, to establish the effectiveness of CLABA, we performed empirical analysis on various synthetic prototype distributed systems. We compare *CALBA* with three other load balancing schemes across multiple performance metrics. In this direction, we performed various experiments with different types of system configurations such as varying system size, varying system utilization, and system heterogeneity. The empirical results discerned that our *CALBA* not only outperforms its competitive algorithms in terms of response time and monetary cost but also yields a commensurable system-wide performance. Finally, in Section 5.4, it is established that *CALBA* is an adaptable and feasible approach to get a cost-aware load balancing solution. Immediate extension to this work can be in two distinct directions. First, as with any computed system, burst arrivals need to be considered. In our work, although we considered the arrivals as a Poisson processes, the results of this work can be directly applied at a higher level by considering burst traffic arrival by a batch arrival process (e.g., bursts of packets arriving according to a Poisson process). However, it would be interesting to model and analyze burst arrival process separately and redesign the strategies proposed in this work. Second, it would be interesting to consider factors such as energy cost and server availability in the formulation which are of practical importance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Avadh Kishor: Conceptualization, Methodology, Writing - original draft, Writing - review & editing. **Rajdeep Niyogi:** Conceptualization, Methodology, Supervision, Writing - review & editing. **Bharadwaj Veeravalli:** Supervision, Writing - review & editing.

Acknowledgments

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. The second author was in part supported by a research grant from Google.

Appendix A

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

A.1. Proof of Theorem 1

According to Lemmas 1 and 2, problem OPT_i involves the minimization of a strict convex function over a convex set. Therefore, first-order *Karush-Kuhn–Tucker* (KKT) conditions are necessary and sufficient for optimality [31]. The Lagrangian associated with NOP is

$$\mathcal{L}(\mathbf{x}_i,\eta,\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_n)=\sum_{j=1}^{n}(w_1+w_2c_j)\frac{x_{ij}}{\mu_j^i-x_{ij}\lambda_i} -\eta\Big(\sum_{j=1}^{n}x_{ij}-1\Big)-\sum_{j=1}^{n}\alpha_j x_{ij}+\sum_{j=1}^{n}\beta_j\Big(x_{ij}\lambda_i-\mu_j^i\Big) \quad (A.1)$$

where $\eta\in\mathbb{R}$ and $\alpha_j\in\mathbb{R},\beta_j\in\mathbb{R}$, are Lagrange multipliers. Suppose, $\mathbf{x}_i=(x_{i1},\ldots,x_{in})$ is the optimal solution of the problem, then KKT conditions for optimality are follows:

$$\frac{\partial\mathcal{L}}{\partial x_{ij}}=0 \quad (A.2)$$

$$\frac{\partial\mathcal{L}}{\partial\eta}=0 \quad (A.3)$$

$$\alpha_j x_{ij}=0,\qquad \forall j\in\mathcal{N} \quad (A.4)$$

$$\beta_j(x_{ij}\lambda_i-\mu_j^i)=0,\qquad \forall j\in\mathcal{N} \quad (A.5)$$

$$\alpha_j\geq 0,\beta_j\geq 0,\qquad \forall j\in\mathcal{N} \quad (A.6)$$

The response time of a server tends to infinity when the process arrival rate on the server is equal to its service rate. Therefore, for each server j, there will always be $(x_{ij}\lambda_i-\mu_j^i)<0$, and (A.5) gives $\beta_j=0$.

These aforementioned KKT conditions become

$$\frac{(w_1+w_2c_j)\mu_j^i}{(\mu_j^i-x_{ij}\lambda_i)^2}-\eta-\alpha_j=0 \quad (A.7)$$

$$\sum_{j=1}^{n}x_{ij}=1 \quad (A.8)$$

$$\alpha_j x_{ij}=0,\quad \forall j\in\mathcal{N} \quad (A.9)$$

$$\alpha_j\geq 0,\quad \forall j\in\mathcal{N} \quad (A.10)$$

We now start by noting that α_j is the only slack variable. So, it can be eliminated leaving

$$x_{ij}\left(\frac{(w_1+w_2c_j)\mu_j^i}{(\mu_j^i-x_{ij}\lambda_i)^2}-\eta\right)=0,\quad \forall j\in\mathcal{N} \quad (A.11)$$

Since, $\alpha_j\geq 0$, we get

$$\eta\leq\left(\frac{(w_1+w_2c_j)\mu_j^i}{(\mu_j^i-x_{ij}\lambda_i)^2}\right),\qquad \forall j\in\mathcal{N} \quad (A.12)$$

For a given server j, if $\eta>((w_1+w_2c_j)/\mu_j^i)$, then (A.12) can only hold if $x_{ij}>0$ as well, which together with (A.11) implies that

$$x_{ij}=\frac{1}{\lambda_i}\left(\mu_j^i-\frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sqrt{\eta}}\right),\text{ if }\eta>\frac{(w_1+w_2c_j)}{\mu_j^i} \quad (A.13)$$

If $\eta\leq((w_1+w_2c_j)/\mu_j^i)$, then $x_{ij}>0$ is impossible, because it would imply that $\eta\leq((w_1+w_2c_j)/\mu_j^i)<(w_1+w_2c_j)\mu_j^i/(\mu_j^i-x_{ij}^*\lambda_i)^2$, which violates the complementary slackness condition (A.11). In conclusion, we have

$$x_{ij}=\begin{cases}\frac{1}{\lambda_i}\left(\mu_j^i-\frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sqrt{\eta}}\right), & \text{if }\eta>\frac{(w_1+w_2c_j)}{\mu_j^i}\\ 0 & \text{otherwise}\end{cases} \quad (A.14)$$

From (A.14), we see that x_{ij} is an increasing function of η. Therefore, there exists a unique value of η as a solution of the (A.8). Since $\frac{w_1+w_2c_j}{\mu_j^i}$ is non-decreasing in j, it now implies that there exists an index l such that

$$\frac{(w_1+w_2c_l)}{\mu_l^i}<\eta<\frac{(w_1+w_2c_{l+1})}{\mu_{l+1}^i} \quad (A.15)$$

from (A.8) and (A.14), we get

$$\eta=\left(\frac{\sum_{j=1}^{l}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{l}\mu_j^i-\lambda_i}\right)^2 \quad (A.16)$$

From (A.15) and (A.16) together, denoting $\varrho=w_1+w_2c_j$, we get

$$l=\sup\left\{k\leq n\colon\frac{(w_1+w_2c_k)}{\mu_k^i}<\left(\frac{\sum_{j=1}^{k}\sqrt{\varrho}\sqrt{\mu_j^i}}{\sum_{j=1}^{k}\mu_j^i-\lambda_i}\right)^2\right\} \quad (A.17)$$

which is the desired result stated in (20).

Substituting (A.17) in (A.14), we get

$$x_{ij}=\begin{cases}\frac{1}{\lambda_i}\left(\mu_j^i-\theta\left(\sum_{j=1}^{l}\mu_j^i-\lambda_i\right)\right), & \text{if } j\in[1,l]\\ 0 & \text{if } j\in[l+1,n]\end{cases} \quad (A.18)$$

where $\theta=\frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{l}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}$. □

A.2. *Proof of Theorem 2*

In steps 4 to 6 of Algorithm 1, a minimum index, $j=k$ such that $\frac{\sqrt{w_1+w_2c_k}}{\mu_k^i}\geq\frac{\sum_{j=1}^{k}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{k}\mu_j^i-\lambda_i}$, is determined. Applying Theorem 1, $x_{ik},x_{ik+1},\ldots,x_{in}$ should all set to zero in order to minimize $f_i(\mathbf{x}_i,\mathbf{x}_{-i})$ (in step 5). Next, in steps 9 to 11, according to Theorem 1, x_{ij} are set to

$$\frac{1}{\lambda_i}\left(\mu_j^i-\frac{\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}{\sum_{j=1}^{k-1}\sqrt{w_1+w_2c_j}\sqrt{\mu_j^i}}\left(\sum_{j=1}^{k-1}\mu_j^i-\lambda_i\right)\right)$$

for $j=[1,k-1]$. Thus, the strategy $\mathbf{x}_i=(x_{i1},x_{i2},\ldots,x_{in})$ computed in Algorithm 1 is guaranteed to be the optimal solution of OPT$_i$ and is also the best response strategy of player i.

References

[1] Y. Jiang, A survey of task allocation and load balancing in distributed systems, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2016) 585–599.

[2] S. Bera, S. Misra, J.J. Rodrigues, Cloud computing applications for smart grid: A survey, IEEE Trans. Parallel Distrib. Syst. (5) (2015) 1477–1494.

[3] W. Rao, L. Chen, A.W.-C. Fu, G. Wang, Optimal resource placement in structured peer-to-peer networks, IEEE Trans. Parallel Distrib. Syst. 21 (7) (2010) 1011–1026.

[4] D. Grosu, A.T. Chronopoulos, Noncooperative load balancing in distributed systems, J. Parallel Distrib. Comput. 65 (9) (2005) 1022–1034.

[5] H. Kameda, E. Altman, Inefficient noncooperation in networking games of common-pool resources, IEEE J. Sel. Areas Commun. 26 (7) (2008).

[6] R. Subrata, A.Y. Zomaya, B. Landfeldt, Game-theoretic approach for load balancing in computational grids, IEEE Trans. Parallel Distrib. Syst. 19 (1) (2008) 66–76.

[7] R. Tripathi, S. Vignesh, V. Tamarapalli, A.T. Chronopoulos, H. Siar, Non-cooperative power and latency aware load balancing in distributed data centers, J. Parallel Distrib. Comput. 107 (2017) 76–86.

[8] S. Song, T. Lv, X. Chen, Load balancing for future internet: An approach based on game theory, J. Appl. Math. 2014 (2014) http://dx.doi.org/10.1155/2014/959782, 959782:1–959782:11.

[9] S. Penmatsa, A.T. Chronopoulos, Game-theoretic static load balancing for distributed systems, J. Parallel Distrib. Comput. 71 (4) (2011) 537–555.

[10] X. Tang, S.T. Chanson, Optimizing static job scheduling in a network of heterogeneous computers, in: International Conference on Parallel Processing, IEEE, 2000, pp. 373–382.
[11] A.Y. Zomaya, Y.-H. Teh, Observations on using genetic algorithms for dynamic load-balancing, IEEE Trans. Parallel Distrib. Syst. 12 (9) (2001) 899–911.
[12] J. Xu, A.Y. Lam, V.O. Li, Chemical reaction optimization for task scheduling in grid computing, IEEE Trans. Parallel Distrib. Syst. 22 (10) (2011) 1624–1631.
[13] Z. Xiao, Z. Tong, K. Li, K. Li, Learning non-cooperative game for load balancing under self-interested distributed environment, Appl. Soft Comput. 52 (2017) 376–386.
[14] 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).
[15] L. Yu, T. Jiang, Y. Zou, Price-sensitivity aware load balancing for geographically distributed internet data centers in smart grid environment, IEEE Trans. Cloud Comput. (2016).
[16] X. Xu, H. Yu, A game theory approach to fair and efficient resource allocation in cloud computing, Math. Probl. Eng. 2014 (2014) 1–14, http://dx.doi.org/10.1155/2014/915878.
[17] S. Penmatsa, A.T. Chronopoulos, Cost minimization in utility computing systems, Concurr. Comput.: Pract. Exper. 26 (1) (2014) 287–307.
[18] Q. Zheng, B. Veeravalli, On the design of mutually aware optimalpricing and load balancing strategies for grid computing systems, IEEE Trans. Comput. 63 (7) (2014) 1802–1811.
[19] V. Jalaparti, G.D. Nguyen, Cloud Resource Allocation Games, Tech. rep., 2010, URL http://hdl.handle.net/2142/17427.
[20] M. Kumar, S. Sharma, Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment, Comput. Electr. Eng. 69 (2018) 395–411.
[21] C. Dastagiraiah, V.K. Reddy, K.V. Pandurangarao, Dynamic load balancing environment in cloud computing based on VM ware off-loading, in: Advances in Intelligent Systems and Computing, Springer Singapore, 2017, pp. 483–492.
[22] I.A. Elgendy, W. Zhang, Y.-C. Tian, K. Li, Resource allocation and computation offloading with data security for mobile edge computing, Future Gener. Comput. Syst. 100 (2019) 531–541.
[23] H. Ghasemi, M.J. Siavoshani, S. Hadadan, A novel communication cost aware load balancing in content delivery networks using honeybee algorithm, 2019, CoRR abs/1902.04463, arXiv:1902.04463.
[24] R. Yadav, W. Zhang, O. Kaiwartya, P.R. Singh, I.A. Elgendy, Y.-C. Tian, Adaptive energy-aware algorithms for minimizing energy consumption and SLA violation in cloud computing, IEEE Access 6 (2018) 55923–55936.
[25] A.C. Zhou, B. Shen, Y. Xiao, S. Ibrahim, B. He, Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters, IEEE Trans. Parallel Distrib. Syst. (2019) 1.
[26] J. Leithon, S. Werner, V. Koivunen, Cost-aware renewable energy management: Centralized vs. distributed generation, Renew. Energy 147 (2020) 1164–1179.
[27] V. Cardellini, M. Colajanni, P.S. Yu, Dynamic load balancing on web-server systems, IEEE Internet Comput. 3 (3) (1999) 28–39.
[28] O. Brun, B. Prabhu, Worst-case analysis of non-cooperative load balancing, Ann. Oper. Res. 239 (2) (2016) 471–495.
[29] B. Yang, Z. Li, S. Chen, T. Wang, K. Li, Stackelberg game approach for energy-aware resource allocation in data centers, IEEE Trans. Parallel Distrib. Syst. 27 (12) (2016) 3646–3658.
[30] H. Zhang, Y. Xiao, S. Bu, R. Yu, D. Niyato, Z. Han, Distributed resource allocation for data center networks: A hierarchical game approach, IEEE Trans. Cloud Comput. (2018).
[31] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, NY, USA, 2004.
[32] A. Beck, Introduction to Nonlinear Optimization -Theory, Algorithms, and Applications with MATLAB, MOS-SIAM Series on Optimization, vol. 19, SIAM, 2014.
[33] J. Soriano, Global minimum point of a convex function, Appl. Math. Comput. (ISSN: 0096-3003) 55 (2) (1993) 213–218.
[34] Y.A. Korilis, A.A. Lazar, A. Orda, Capacity allocation under noncooperative routing, IEEE Trans. Automat. Control 42 (3) (1997) 309–325.
[35] A. Orda, R. Rom, N. Shimkin, Competitive routing in multiuser communication networks, IEEE/ACM Trans. Netw. (ToN) 1 (5) (1993) 510–521.
[36] R. Jain, D.-M. Chiu, W. Hawe, A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems, Technical Report DEC-TR-301, Tech. Rep., Digital Equipment Corporation, 1984.
[37] O. Pearce, T. Gamblin, B.R. De Supinski, M. Schulz, N.M. Amato, Quantifying the effectiveness of load balance algorithms, in: Proceedings of the 26th ACM International Conference on Supercomputing, ACM, 2012, pp. 185–194.

Avadh Kishor is doing Ph.D. in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Roorkee, India. His research interest include distributed systems, multi-agent systems and algorithmic game theory. He did M.Tech. in Computer Science and Engineering from Indian Institute of Information Technology and Management (IIITM) Gwalior, India, in 2015. He is a member of ACM and IEEE.

Rajdeep Niyogi is an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Roorkee, India. His research interests include automated planning, algorithmic game theory, distributed systems, and applications of logic and automata theory. He did Ph.D. in Computer Science and Engineering from Indian Institute of Technology (IIT) Kharagpur, India, in 2004. He is a member of ACM and ERCIM.

Bharadwaj Veeravalli is a Senior Member, IEEE & IEEE-CS,received the B.Sc. degree in physics, from Madurai-Kamaraj University, India, in 1987, the Master's degree in Electrical Communication Engineering from the Indian Institute of Science(IISc), Bangalore, India in 1991, and the Ph.D. degree from the Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He received gold medals for his bachelor degree overall performance and for an outstanding Ph.D. thesis (IISc, Bangalore India) in the years 1987 and 1994, respectively. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) division, at The National University of Singapore, Singapore, as a tenured Associate Professor. His main stream research interests include cloud/grid/cluster computing (big data processing, analytics and resource allocation), scheduling in parallel and distributed systems, Cybersecurity, and multimedia computing. He is one of the earliest researchers in the field of Divisible Load Theory (DLT). He is currently serving the editorial board of IEEE Transactions on Parallel & Distributed Systems as an Associate Editor. He is a senior member of the IEEE and the IEEE-CS. He can be contacted via: elebv@nus.edu.sg