A Game Approach to Multi-Servers Load Balancing with Load-Dependent Server Availability Consideration

Chubo Liu, Kenli Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—In this paper, we focus on request migration strategies among multi-servers for load balancing. Different from the general load balancing problem, we consider it under a distributed, non-cooperative, and competitive environment. Due to the mentioned characteristics, we view our problem from a game theoretic perspective and formulate it into a non-cooperative game among the multiple servers, in which each server is informed with incomplete information of other servers. For each server, we define its expected response time as a disutility function and try to minimize its value. We also take into account server availability, which impacts the processing capacity of a server and thus its disutility. We solve the problem by employing variational inequality (VI) theory and prove that there exists a Nash equilibrium solution set for the formulated game. Then, we propose an iterative proximal algorithm (IPA) to compute a Nash equilibrium solution. The convergence of the IPA algorithm is also analyzed and we find that it converges to a Nash equilibrium. Finally, we conduct some numerical calculations to verify our theoretical analyses. The experimental results show that our proposed IPA algorithm converges to a Nash equilibrium very quickly and significantly decreases the disutilities of all servers by configuring a proper request migration strategy.

Index Terms—Distributed environment, load balancing, non-cooperative game, nash equilibrium, server availability, variational inequality theory

1 Introduction

1.1 Motivation

Cloud computing is the delivery of resources and computing as a service rather than a product over the Internet, such that accesses to shared hardware, software, databases, information, and all resources are provided to consumers on-demand [1], [2]. Usually, the provided services mainly refer to Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), which can all be accessed by the general public on demand [3]. However, due to the more and more users, the portal server of a cloud center can be heavily loaded. Furthermore, many lightweight jobs can bring a lot of interferences to other computation-intensive jobs and thus significantly degrade their performances. To improve the overall service quality (e.g., response time) of a cloud center, recently, some works (such as [4]) propose to deploy certain edge servers in front of a cloud center. As shown in Fig. 1, user applications (demands) first arrive at edge servers, which are interconnected through a network. If an application is lightweight, then it is immediately processed by one of the edge servers. Otherwise, it is uploaded to the cloud center for processing. Obviously, this can mitigate the interferences brought by lightweight jobs and significantly improve the overall performance of a cloud center.

In this work, we focus on load balancing for edge servers layer. Due to the differences in the computing capacities and uneven request arrival patterns, the workload on different servers can vary greatly [5]. It can often be seen that many servers are underutilized while the others are overloaded. Or on the other hand, the cloud providers do not take into account server availability, which refers to the percentage of time that it is running in a given time interval [6]. All of these situations can lead to high response time and low service availability, which are two import quality measures for a cloud provider to appeal more users to use cloud service [7]. Therefore, it is important for a cloud provider to design an appropriate load balancing scheme involving server availability. In this work, we try to configure a proper request migration strategy for the edge servers layer. Specifically, we try to simultaneously optimize the response times of all servers involved in the connected multi-server system. In addition, we take server availability into account.

Load balancing concerns the distribution of requests among diverse servers in a given environment such that no server is overloaded or underutilized [8], [9]. It is one of the most important factors that should be seriously considered by both a cloud provider and its multiple users. For a cloud provider, an appropriate load balancing strategy helps in making use of the available resources most favourably and thus ensures that no server is overloaded or underutilized.

- *C. Liu and K. Li are with the College of Information Science and Engineering, and National Supercomputing Center in Changsha, Hunan University, Hunan 410082, China. E-mail: {liuchubo, lkl}@hnu.edu.cn.*
- *K. Li is with the College of Information Science and Engineering, and National Supercomputing Center in Changsha, Hunan University, Hunan 410082, China, and with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 5 Apr. 2017; revised 1 Oct. 2017; accepted 1 Jan. 2018. Date of publication 15 Jan. 2018; date of current version 5 Mar. 2021. (Corresponding author: Kenli Li.)
Recommended for acceptance by G. Fox.
Digital Object Identifier no. 10.1109/TCC.2018.2790404

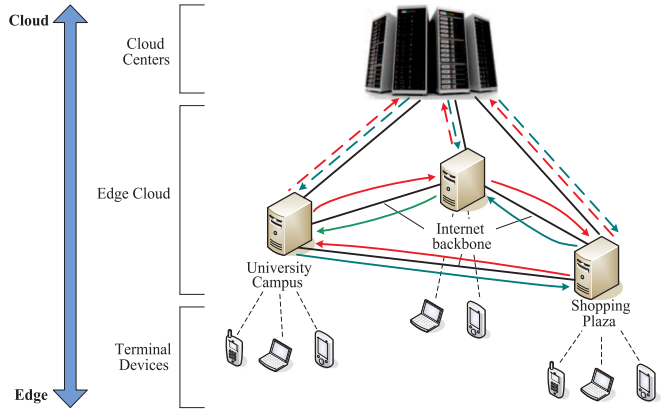


Fig. 1. Motivation illustration.

It can improve the aggregated performance and increase the revenue of the cloud provider. Specifically, an appropriate load balancing scheme improves the request service rate of the cloud provider. Therefore, the cloud provider can charge more from its users for more accomplished work and thus increase its revenue. Cloud-based applications depend even more heavily on load balancing and optimization than traditional enterprise applications [1]. On the other hand, it improves service quality (e.g., task response time) and appeals more users to use cloud service. For multiple cloud users, appropriate load balancing will be seriously considered when they select a cloud provider. The same task, such as running an online voice recognition algorithm, is able to generate more utility for a cloud user if it can be completed within a shorter period of time [10].

Server availability refers to the probability that a server is found to be in the running state at a given point in time. It can also be defined as the percentage of time that it is running in a given time interval [6]. Server availability is an important factor that should be taken into account by a cloud provider for designing appropriate load balancing scheme. The reason behind lies in that server availability is also related to the profit of a cloud provider and the appeals to more cloud users in the market. Specifically, if availability level of servers is low, it impacts the service availability of servers and thus decreases the total processing rate of the cloud provider. Obviously, the cloud provider charges less from cloud users due to its less accomplished work during certain time interval. On the other hand, low server availability also impacts the service quality (e.g., task response time) of the cloud provider, which is seriously considered by most cloud users. When more requests are allocated to the servers with low availability, even though their ideal processing capacities are high, the overall response time can be long and thus dissatisfies its current users even the potential users in the market. Therefore, it is important for a cloud provider to design appropriate load balancing scheme which also involves server availability.

1.2 Related Work

Load balancing is one of the most important factors that should be taken into account in scheduling, and in literature, many works have been done on it for various considerations [1], [5], [11], [12], [13], [14], [15].

Specifically, in [1], Cao et al. studied the tradeoff between an aggregated performance (system response time) and energy consumption in cloud centers. Specifically, the tried to optimize system response time under energy constraint and energy consumption under response time constraint, respectively. In [11], KASSAB et al. addressed load balancing (makespan) for scheduling independent tasks under power constraints. They proposed several heuristics for the case of multi-core architecture and assessed their performances on synthetic workloads and power envelopes. In [12], Zhao et al. proposed a Bayes theorem based heuristic, and tried to optimize the standard deviation to achieve load balancing. However, as shown in Table 1, all these works refer to one single objective (makespan, overall system response time, or standard deviation of response time), i.e., they focus on load balancing metric.

There are also some works referring multi-objectives which incorporate load balancing, i.e., load balancing is one of the multiple objectives. Siavoshani et al. [13] combined load balancing (average waiting time) with communication cost and tried to configure a scheduling scheme by optimizing a weighted sum of them. In [15], Shen studied load balancing by migrating virtual machines (VMs), and she aimed to optimize the number of overall VM migrations. She also incorporated communication cost, bandwidth, and proposed a heuristic to try to optimize all of them. In [14], Thant et al. proposed a heuristic for the purposes of minimizing the overall makespan and machine instance deployment cost during workload execution. For this kind of scheduling, the main techniques are usually heuristic and weighted sum method (see Table 1). However, as shown in Table 1, all the previously mentioned works are from the central perspective. Different from their considerations, we address load balancing under distributed environment.

When considering load balancing under distributed environment, the problem becomes more complex. The reason behind lies in that the relationships among servers are

TABLE 1
Load Balancing Comparison Between IPA and the State-of-the-Art Schemes

Schemes	Environment	Objective(s)	Main technique(s)
[11]	Central	Single objective (makespan)	Heuristic
[12]	Central	Single objective (standard deviation)	Heuristic
[1]	Central	Single objective (system response time)	Lagrangian multiplier method
[13]	Central	Multi-objectives ($= 2$)	Weighted sum method
[14]	Central	Multi-objectives ($= 2$)	Heuristic
[15]	Central	Multi-objectives ($= 3$)	Heuristic
[5]	Distributed	Multi-objectives (≥ 3)	Non-cooperative game
IPA	Distributed	Multi-objectives (≥ 3)	Non-cooperative game and variational inequality theory

non-cooperative and competitive. Few works can be found for this in the literature. In [5], Penmatsa and Chronopoulos proposed a non-cooperative game based NCOOPC algorithm, in which they tried to minimize the average response time of each cloud user. In this work, we also use non-cooperative game to formulate our problem. However, due to the different models and problems. The general game methods (such as NCOOPC in [5]) can no longer be applied in our situation. In this work, we leverage variational inequality (VI) theory to solve our problem.

Game theory is a field of applied mathematics that describes and analyzes scenarios with interactive decisions [16], [17]. It is a formal study of conflicts and cooperation among multiple competitive players [18], and a powerful tool for design and control of multiagent systems [19]. There has been a growing interest in adopting cooperative and non-cooperative game approaches to modeling many problems [20], [21], [22]. In [20], the authors presented a game-theoretic approach for the provisioning and operation of the infrastructure under uniform cost models. Xu and Yu [21] proposed a game theoretic resource allocation algorithm which considers the fairness among cloud users and resource utilization. However, to obtain game solutions, there are not standard methods, and we must address game problems on a case by case basis. In this work, we try to leverage variational inequality theory, which is a framework suitable for investigating and solving various equilibrium models and optimization problems in nonlinear analysis [16], [23]. We try to obtain a Nash equilibrium for our non-cooperative and competitive load balancing. For more works on game theory, the reader is referred to [24], [25], [26], [27], [28].

Server availability is another important metric in scheduling. It refers to the percentage of time that it is running in a given time interval [6]. Obviously, it impacts the actual processing rates of servers and thus the aggregated performance in cloud.

During the past decades, the vast majority of researches in load balancing assume that all servers are continuously available for processing throughout horizon [29]. This assumption may not be true in all cases, since a server may become unavailable due to breakdown, preventive maintenance, tool change during a certain period [30], [31], [32]. Some works have been done to investigate load balancing schemes with availability constraints [29], [33]. In [29], Qin and Xie proposed an algorithm SSAC, in which they tried to maximize system availability while minimizing average response time at the same time. In [33], Liao and Sheen studied a polynomial time binary search algorithm to solve the parallel server scheduling problem with availability constraints. However, rare of above listed researches have considered the impacts of workload on server availability. Nevertheless, according to surveys in [6], [34], [35], a server is more likely to fail if it is heavily loaded.

Therefore, we also involve load-dependent server availability in our non-cooperative and competitive load balancing scheme.

1.3 Our Contributions

In this paper, we focus on request migration strategies on multi-servers for load balance. Different from the general load balancing problem, we consider it under a distributed, non-cooperative, and competitive environment. The goal is to minimize the average response time for each server as much as possible. The main contributions and differences of this paper are listed as follows.

- We consider load balancing under a distributed and non-cooperative environment. Furthermore, we try to find a request migration strategy which simultaneously optimizes the expected response times of all servers rather than a single objective (such as makespan).
- We consider the problem from a game theoretic perspective and formulate it into a non-cooperative game among the multiple servers. Specifically, each server is informed with incomplete information of other servers. We define its average response time as the disutility function and try to minimize its value.
- We also take into account server availability, which impacts the processing capacity of a server and thus its disutility function value.
- We solve the problem by employing variational inequality theory and prove that there exists a Nash equilibrium solution set for the formulated game. Then, we propose an iterative proximal algorithm (IPA) to compute a Nash equilibrium solution.
- The convergence of the IPA algorithm is also analyzed and we find that it converges to a Nash equilibrium.

We conduct some numerical calculations to verify our theoretical analysis. The experimental results show that our proposed IPA algorithm converges to a Nash equilibrium very quickly and significantly decreases the disutilities of all servers by configuring a proper request migration strategy.

1.4 Organization

The rest of the paper is organized as follows. Section 2 describes the system models of the system and presents the problem to be solved. Section 3 formulates the problem into a non-cooperative game and solves the problem by employing variational inequality theory. Some analyses are also presented in this section. Section 4 is developed to verify our theoretical analyses and show the effectiveness of our proposed algorithm. We conclude the paper with future work in Section 5.

2 System Model and Problem Formulation

To begin with, we present our system model in the context of a cloud provider with m heterogeneous servers, which are connected by a communication network. We denote the set of servers as $\mathcal{M} = \{1, \ldots, m\}$. The processing capacity of server i ($i \in \mathcal{M}$) is represented by its service rate μ_i, which is also impacted by the workload on the server. The external request arrival rate at server i ($i \in \mathcal{M}$) is assumed to follow a Poisson process. Similar to [5], [36], we also assume that each of the servers and the communication network are modeled as M/M/1 queuing systems.

We summarize all the notations used in this section in the notation Table 2.

2.1 Request Migration Model

We consider the request migration model motivated by [5], [36], where the request migration profile at server i ($i \in \mathcal{M}$)

TABLE 2
Notations

Notation	Description
m	Number of servers in the cloud center
$\mathcal{M}$	Set of the m servers in the cloud center
p_{ij}	Probability of a request transferred from server i to j
$\boldsymbol{p}_i$	Probability vector for transferring requests at server i
$\mathcal{P}_i$	Probability strategy set at server i
λ_{ij}	Request migration rate from server i to j
$\boldsymbol{\lambda}_i$	Request migration vector at server i
$\boldsymbol{\lambda}_\Sigma$	Summation of $\boldsymbol{\lambda}_i$ for all $i \in \mathcal{M}$
$\boldsymbol{\lambda}$	Request migration strategies of all servers
$\boldsymbol{\lambda}_{-i}$	Migration strategies of all servers except server i
$\mathcal{Q}_i$	Request migration strategy set at server i
$\mathcal{Q}$	Joint request migration strategy set of all servers
ϕ_i	External request arrival rate at server i
β_i	Aggregated request arrival rate at server i
μ_i	Expected processing capacity of server i
$\bar{\mu}_i$	Processing capacity of server i without load impacts
$\boldsymbol{\mu}$	Processing vector involving $\bar{\mu}_i$ for all $i \in \mathcal{M}$
α_i	Availability deteriorating factor of server i
$\tilde{D}_i$	Delay of a request incurred at server i
D_i	Expected delay of a request incurred at server i
t	Mean communication time of a request
γ	Total traffic through the network
$\tilde{C}$	Communication delay incurred by a request
C	Expected communication delay incurred by a request
$\tilde{R}_{ij}$	Delay of a request at i and processed at j
R_{ij}	Expected delay of a request at i and processed at j
F	Probability density function of $\tilde{R}_{ij}$
$\tilde{R}_i$	Response time of a request at server i
R_i	Expected response time of a request at server i
U_i	Disutility function of server i
$\mathbf{U}$	Disutility function vector for all servers

is formulated as the following probability vector

$$\boldsymbol{p}_i = (p_{i1}, \ldots, p_{im})^T, \tag{1}$$

where p_{ij} ($j \in \mathcal{M}$) is the probability that a request at server i is migrated to server j and it is subject to the constraint $\sum_{j=1}^{m} p_{ij} = 1$. Then, the request migration strategy set of server i ($i \in \mathcal{M}$) can be expressed as

$$\mathcal{P}_i = \left\{ \boldsymbol{p}_i \middle| \sum_{j=1}^{m} p_{ij} = 1 \text{ and } p_{ij} \geq 0,\ \forall j \in \mathcal{M} \right\}. \tag{2}$$

At each server, there are randomly arrived external requests. Denote ϕ_i as the external request arrival rate at server i ($i \in \mathcal{M}$), and λ_{ij} as the request flow rate from server i to server j (i.e., the expected number of requests sent from server i to server j per unit of time). Then

$$\lambda_{ij} = p_{ij}\phi_i, \tag{3}$$

and the request migration profile can also be expressed as

$$\boldsymbol{\lambda}_i = (\lambda_{i1}, \ldots, \lambda_{im})^T. \tag{4}$$

Correspondingly, the request migration strategy set can be written as

$$\mathcal{Q}_i = \left\{ \boldsymbol{\lambda}_i \middle| \sum_{j=1}^{m} \lambda_{ij} = \phi_i \text{ and } \lambda_{ij} \geq 0,\ \forall j \in \mathcal{M} \right\}, \tag{5}$$

and the joint individual strategy set of all servers is given as $\mathcal{Q} = \mathcal{Q}_1 \times \cdots \times \mathcal{Q}_m$.

Denote μ_j as the processing rate of server j ($j \in \mathcal{M}$). Then, the aggregated requests at each server cannot exceed its processing capacity, i.e., $\beta_j = \sum_{i=1}^{m} \lambda_{ij} < \mu_j$, where β_j is the workload of server j ($j \in \mathcal{M}$). Then, when given the others' request migration strategies $\boldsymbol{\lambda}_{-i} = (\boldsymbol{\lambda}_j)_{j=1, j\neq i}^{m}$, the request migration strategy $\boldsymbol{\lambda}_i$ of server i ($i \in \mathcal{M}$) also satisfies $\sum_{i=1}^{m} \lambda_{ij} < \mu_j$.

2.2 Server Availability Model

Server availability refers to the probability that a server is found to be in the running state at a given point in time [6]. It can also be defined as the percentage of time that it is running in a given time interval. A relationship between the average load intensity and the failure rate does exist [6], [35]. If the workload intensity increases, the more failures will be expected.

To take server availability into account for appropriate load balance, we model the server availability in our work motivated by [6], where the processing capacity (i.e., the available service time) of a server linearly decreases with the increase of its workload. Specifically, the expected service rate of server i ($i \in \mathcal{M}$) is given as follows:

$$\mu_i = \bar{\mu}_i - \alpha_i \beta_i, \tag{6}$$

where $\bar{\mu}_i$ is the maximum processing rate of server i, α_i ($\alpha_i < 1$) is the corresponding availability deteriorating factor, and β_i is its workload (i.e., the aggregated requests at server i), which satisfies $\beta_i < \bar{\beta}_i$ with $\bar{\beta}_i = \frac{\bar{\mu}_i}{1+\alpha_i}$. In this work, we assume that the total external request arrival rate is less than the total expected processing rate of all servers, i.e., $\sum_{i=1}^{m} \phi_i < \sum_{i=1}^{m} \bar{\beta}_i$.

2.3 Cloud Service Model

There are m heterogeneous servers, which are connected by a communication network. Each of the servers is modeled as an M/M/1 queuing system, serving the requests aggregated at this server.

A request at server i ($i \in \mathcal{M}$) may be either processed at server i or transferred to another server j ($j \in \mathcal{M}$) through the communication network for remote processing. The response time of a request in above system consists of queuing delay and processing delay at the service server, and also some possible communication delay incurred due to request transfer. Denote $\tilde{D}_i(\beta_i)$ as the delay incurred at the server i ($i \in \mathcal{M}$), then the probability that $\tilde{D}_i(\beta_i)$ is greater than T ($T \geq 0$) is equal to $e^{-(\mu_i - \beta_i)T}$, i.e.,

$$\mathbf{Pr}\big[\tilde{D}_i(\beta_i) > T\big] = e^{-(\mu_i - \beta_i)T},\ T \geq 0, \tag{7}$$

and the expected delay is expressed as

$$D_i(\beta_i) = \frac{1}{\mu_i - \beta_i}, \tag{8}$$

where β_i is the aggregated requests at server i, i.e., $\beta_i = \sum_{j=1}^{m} \lambda_{ji}$, and μ_i is the service rate of server i as in (6).

As mentioned earlier, the communication network is also modeled as an M/M/1 queuing system [5], [36], in which the expected communication delay from server i to server j

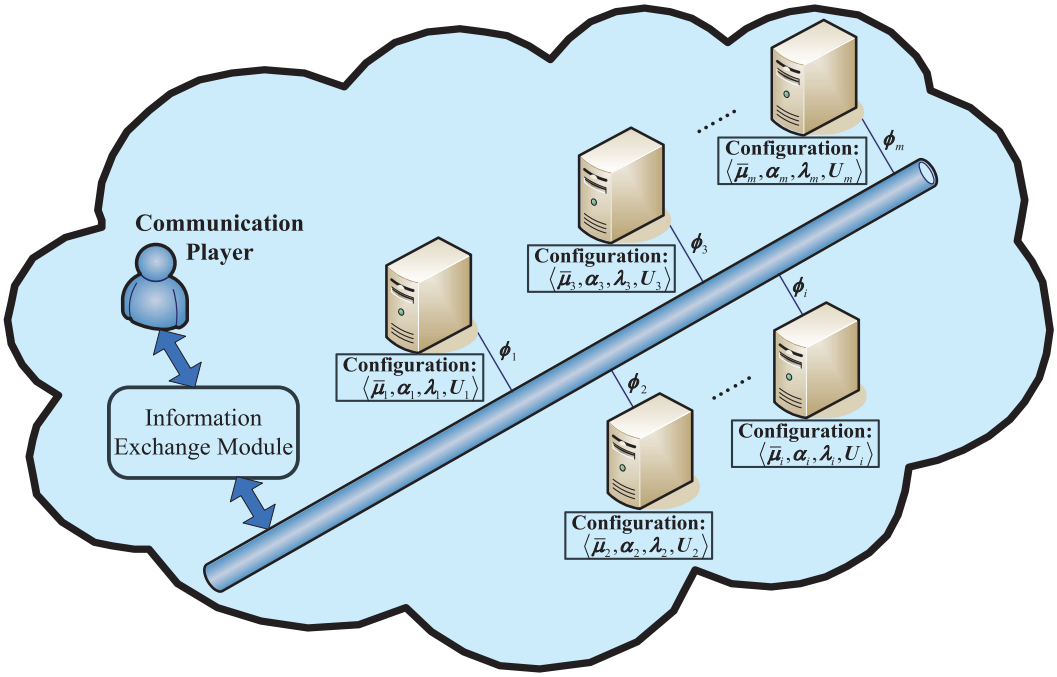


Fig. 2. Architecture model.

is independent of the source-destination pair (i, j) but may depend on the total traffic through the network denoted by γ, where $\gamma = \sum_{i=1}^{m} \sum_{j=1, j\neq i}^{m} \lambda_{ij}$. Denote $\tilde{C}(\gamma)$ as the communication delay for one request and $C(\gamma)$ as the expected value, respectively. Then, with referring to [5], [36], we obtain

$$\mathbf{Pr}\big[\tilde{C}(\gamma) > T\big] = e^{-\left(\frac{1}{t}-\gamma\right)T}, \ T \geq 0, \tag{9}$$

and

$$C(\gamma) = \frac{t}{1 - t\gamma}, \tag{10}$$

where t ($t < \frac{1}{\gamma}$) is the mean communication time for sending or receiving a request.

Denote $\tilde{R}_{ij}(\boldsymbol{\lambda})$ as the delay of a request incurred at server i and been processed at server j ($i, j \in \mathcal{M}$). Then

$$\tilde{R}_{ij}(\boldsymbol{\lambda}) = \begin{cases} \tilde{D}_j(\beta_j) + \tilde{C}(\gamma), & \text{if } j \neq i; \\ \tilde{D}_j(\beta_j), & \text{if } j = i, \end{cases} \tag{11}$$

and

$$\begin{aligned} \mathbf{Pr}\big[\tilde{D}_j + \tilde{C} > T\big] &= \mathbf{Pr}\big[\tilde{D}_j > T\big] \\ &\quad + \int_0^T \int_{T-\tilde{D}_j}^{\infty} \alpha_j e^{-\alpha_j \tilde{D}_j} \alpha_{\tilde{C}} e^{-\alpha_{\tilde{C}} \tilde{C}} d\tilde{D}_j d\tilde{C} \\ &= \frac{\alpha_j}{\alpha_j - \alpha_{\tilde{C}}} e^{-\alpha_{\tilde{C}} T} - \frac{\alpha_{\tilde{C}}}{\alpha_j - \alpha_{\tilde{C}}} e^{-\alpha_j T}, \end{aligned} \tag{12}$$

where $\alpha_j = \mu_j - \beta_j$ and $\alpha_{\tilde{C}} = \frac{1}{t} - \gamma$. We can further obtain its probability density function as

$$F(T) = \frac{\partial}{\partial T} \mathbf{Pr}\big[\tilde{D}_j + \tilde{C} \leq T\big] = \frac{\alpha_j \alpha_{\tilde{C}}}{\alpha_j - \alpha_{\tilde{C}}} \left(e^{-\alpha_{\tilde{C}} T} - e^{-\alpha_j T}\right), \tag{13}$$

and the expected value of $\tilde{R}_{ij}(\boldsymbol{\lambda})$ as

$$R_{ij}(\boldsymbol{\lambda}) = \begin{cases} D_j(\beta_j) + C(\gamma), & \text{if } j \neq i; \\ D_j(\beta_j), & \text{if } j = i. \end{cases} \tag{14}$$

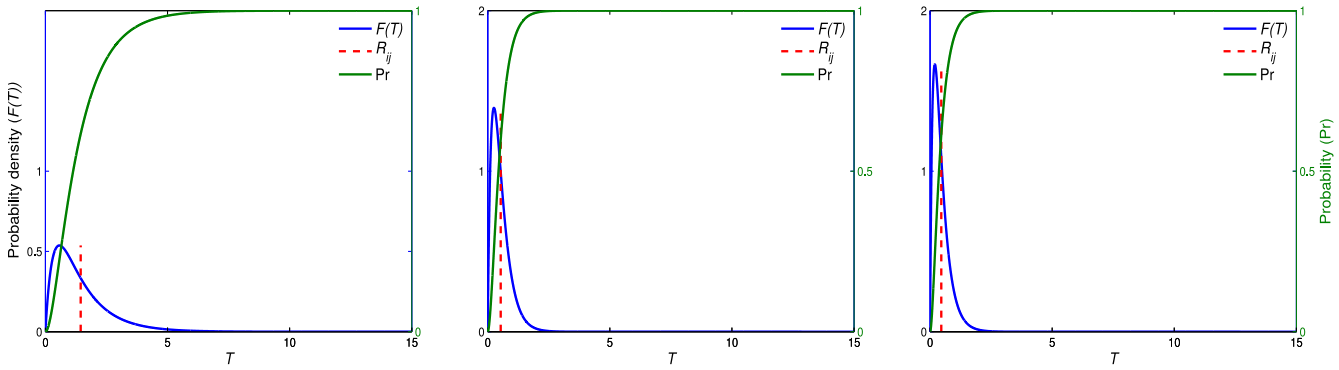
2.4 Architecture Model

In this section, we model the architecture of our proposed framework for load balance in cloud with load dependent server availability. Each of the servers can make appropriate request migration decision through the information exchange model connected by a virtual communication player. As shown in Fig. 2, each server i ($i \in \mathcal{M}$) is equipped with a maximum processing capacity ($\bar{\mu}_i$) with corresponding availability deteriorating rate (α_i), a disutility function (U_i), and the request migration strategy (λ_i). Let $\boldsymbol{\lambda}_\Sigma$ be the aggregated request vector on all servers, then we have $\boldsymbol{\lambda}_\Sigma = \sum_{i=1}^{m} \boldsymbol{\lambda}_i$. Denote $\bar{\boldsymbol{\mu}} = (\bar{\mu}_i)_{i\in\mathcal{M}}$ as the maximum processing capacity vector of all servers, $\boldsymbol{\alpha} = (\alpha_i)_{i\in\mathcal{M}}$ as the corresponding deteriorating rates, and $\mathbf{U} = (U_i)_{i\in\mathcal{M}}$ as the disutility functions of all servers. The cloud provider consists of m heterogeneous servers, which are connected by a communication network, and a virtual communication player, that is, the player can be a daemon running on a host rather than a real player. It communicates some information (e.g., current aggregated request vector on all servers $\boldsymbol{\lambda}_\Sigma$, and the communication requests through the network γ) with multiple servers through the information exchange module. When multiple servers try to make request migration decisions, they first get information from the communication player by the information exchange module, then configure appropriate request migration strategies ($\boldsymbol{\lambda}$) such that their own disutilities ($\mathbf{U}$) are minimized. After this, they send the updated strategies to the communication player. The procedure is terminated when the request migration strategies of all servers are kept fixed.

2.5 Problem Formulation

Denote $\tilde{R}_i$ as the response time of a request at server i ($i \in \mathcal{M}$), i.e., the time interval of a request from its arrival at server i until its service completion. Then, $\tilde{R}_i = \tilde{R}_{ij}$, if the request is allocated to server $j \in \mathcal{M}$.

In this work, we try to minimize the response time ($\tilde{R}_i$) of each server for load balance. Specifically, given a cloud



(a) $\alpha_j = 3$, $\alpha_{\tilde{C}} = 0.9$ (b) $\alpha_j = 3$, $\alpha_{\tilde{C}} = 5$ (c) $\alpha_j = 3$, $\alpha_{\tilde{C}} = 8$

Fig. 3. Illustration for response time.

provider with m heterogeneous servers, denoted by $\mathcal{M} = \{1, \ldots, m\}$, which are connected by a communication network, and the external request arrival rate ϕ_i at server i ($i \in \mathcal{M}$), and the maximum service rate $\bar{\mu}_i$ with availability deteriorating factor α_i ($i \in \mathcal{M}$), we try to find a migration strategy profile for each of the servers such that its response time is minimized.

Notice that, a request at server i ($i \in \mathcal{M}$) is randomly migrated to server j ($j \in \mathcal{M}$) according to a request migration profile $\boldsymbol{p}_i$. Hence, we configure its expectation

$$R_i(\boldsymbol{\lambda}) = \sum_{j=1}^{m} p_{ij} \tilde{R}_{ij}(\boldsymbol{\lambda}), \tag{15}$$

and try to minimize its value. We may further notice that the value of $\tilde{R}_{ij}$ is also random and there is not an equation form to represent it. However, we depict its density function and probability function, and find that $\tilde{R}_{ij}$ does not deviate its average value (R_{ij}) enough (see Fig. 3). Hence, we use R_{ij} to replace $\tilde{R}_{ij}$ and obtain

$$\begin{aligned} R_i(\boldsymbol{\lambda}) &= \sum_{j=1}^{m} p_{ij} R_{ij}(\boldsymbol{\lambda}) \\ &= \sum_{j=1}^{m} p_{ij} D_j(\beta_j) + \sum_{j=1, j\neq i}^{m} p_{ij} C(\gamma) \\ &= \frac{1}{\phi_i} \left(\sum_{j=1}^{m} \frac{\lambda_{ij}}{\mu_j - \beta_j} + \sum_{j=1, j\neq i}^{m} \frac{\lambda_{ij} t}{1 - t\gamma} \right). \end{aligned} \tag{16}$$

Our goal is to find a request migration strategy such that the expected response times of all servers can be optimized, i.e., we try to find a solution to the following non-linear optimization problem (OPT_i): $\forall i \in \mathcal{M}$

$$\begin{aligned} \text{minimize} \quad & R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}),\ \boldsymbol{\lambda}_i \in \mathcal{Q}_i, \\ \text{s.t.} \quad & \lambda_{ij} + \Lambda_{-i}^{j} < \bar{\beta}_j,\ \forall j \in \mathcal{M}, \end{aligned} \tag{17}$$

where $\boldsymbol{\lambda}_{-i} = (\boldsymbol{\lambda}_j)_{j=1, j\neq i}^{m}$ denotes the request migration strategy of all other servers except that of server i, $\bar{\beta}_j = \frac{\bar{\mu}_j}{1+\alpha_j}$, and $\Lambda_{-i}^{j} = \sum_{l=1, l\neq i}^{m} \lambda_{lj}$ for all $j \in \mathcal{M}$.

Remark 2.1. In finding solution to (OPT_i), the request migration strategies of other servers are kept fixed. So the variable in (OPT_i) is the request migration strategy of server i, i.e., $\boldsymbol{\lambda}_i$.

3 Game Formulation and Analyses

In this section, we formulate the request migration problem among multiple servers as a non-cooperative game. By employing variational inequality theory, we analyze the existence of Nash equilibrium solution set for the formulated game. Then, we propose an iterative proximal algorithm to compute a Nash equilibrium solution. We also analyze the convergence of the proposed algorithm.

3.1 Game Formulation

Game theory studies the problems in which players try to maximize their utilities or minimize their disutilities. As described in [5], a non-cooperative game consists of a set of players, a set of disutility functions, and a set of strategies. In this paper, each server is regarded as a player, i.e., the set of players is the m servers. The disutility function of each player i ($i \in \mathcal{M}$) is its average response time, i.e., R_i. The individual strategy set of player i is the request migration set of server i, i.e., $\mathcal{Q}_i$. Then, the joint individual strategy set of all players is given as $\mathcal{Q} = \mathcal{Q}_1 \times \cdots \times \mathcal{Q}_m$.

In our work, we ignore the constraint in (17), because there exists one optimal solution to the optimization problem (17) without violating the constraint in (17), when given the feasible strategies of others $\boldsymbol{\lambda}_{-i} = (\boldsymbol{\lambda}_j)_{j=1, j\neq i}^{m}$. The details are discussed in the following theorem.

Theorem 3.1. *Given two servers $j, k \in \mathcal{M}$ such that their remain processing capacities $\hat{\mu}_j < \hat{\mu}_k$ with $\hat{\mu}_l = \bar{\mu}_l - (1+\alpha_l)\Lambda_{-i}^{l}$ ($l = j, k$), there exists an arrival rate $\phi_i^* > 0$ such that it is optimal to migrate all requests to server k for all requests $\phi_i < \phi_i^*$. Otherwise, it is optimal to migrate requests to both servers.*

Proof. To improve the overall readability of this manuscript, the complete proof of this theorem is given in the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCC.2018.2790404. □

Theorem 3.2. *Reorder the servers in $\mathcal{M}$ such that they are numbered in non-increasing order of their remaining processing rates, i.e., $\hat{\mu}_1 \leq \hat{\mu}_2 \leq \cdots \leq \hat{\mu}_m$ with $\hat{\mu}_j = \bar{\mu}_j - (1+\alpha_j)\Lambda_{-i}^{j}$ ($j \in \mathcal{M}$). There exists a set of threshold request arrival rates $\phi_{i1}^{\min}, \phi_{i2}^{\min}, \ldots, \phi_{im}^{\min} = 0$, such that in the optimal migration policy $\lambda_{ij} = 0$ if $\phi_i \leq \phi_{ij}^{\min}$ and $\lambda_{ij} > 0$ if $\phi_i > \phi_{ij}^{\min}$.*

Proof. To improve the overall readability of this manuscript, the complete proof of this theorem is given in the supplementary material, available online. □

Based on the results of Theorems 3.1 and 3.2, we conclude that there exists one optimal solution to the optimization problem (17) without violating the constraint in (17), when given the feasible strategies of others $\boldsymbol{\lambda}_{-i} = (\boldsymbol{\lambda}_j)_{j=1,j\neq i}^{m}$. Therefore, in our work, we ignore the constraint in (17), and reduce the problem (17) to the following optimization problem:

$$\text{minimize} \quad R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}),\ \boldsymbol{\lambda}_i \in \mathcal{Q}_i. \tag{18}$$

The above formulated game can be formally defined by the tuple $G = \langle \mathcal{Q}, \mathbf{R} \rangle$, where $\mathbf{R} = (R_i)_{i=1}^{m}$. The aim of player i ($i \in \mathcal{M}$), given the other players' strategies $\boldsymbol{\lambda}_{-i}$, is to choose a strategy $\boldsymbol{\lambda}_i \in \mathcal{Q}_i$ such that its average response time $R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})$ is minimized.

Definition 3.1 (Nash equilibrium). *A Nash equilibrium of the formulated game $G = \langle \mathcal{Q}, \mathbf{R} \rangle$ defined above is a request migration profile $\boldsymbol{\lambda}^*$ such that for every player i ($i \in \mathcal{M}$)*

$$\boldsymbol{\lambda}_i^* \in \arg\min_{\boldsymbol{\lambda}_i \in \mathcal{Q}_i} R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}),\ \boldsymbol{\lambda}^* \in \mathcal{Q}. \tag{19}$$

At the Nash equilibrium, each player cannot further decrease its average response time by choosing a different request migration strategy while the strategies of other players are fixed. The equilibrium strategy profile can be found when each player's strategy is the best response to the strategies of other players.

3.2 Nash Equilibrium Existence Analysis

In this section, we analyze the existence of Nash equilibrium solution for the formulated game $G = \langle \mathcal{Q}, \mathbf{R} \rangle$, and prove the existence problem by employing variational inequality theory. Before addressing the problem, we first show three import properties presented in Theorems 3.1, 3.2, and 3.3, which are helpful to prove the existence of Nash equilibrium for the formulated game.

Theorem 3.3. *If both matrixes $\mathcal{M}_1$ and $\mathcal{M}_2$ are positive definite, then the matrix $\mathcal{M}_3$ and $\mathcal{M}_4$ and are also positive definite, where*

$$\mathcal{M}_3 = \mathcal{M}_1 + \mathcal{M}_2, \text{ and } \mathcal{M}_4 = \begin{bmatrix} \mathcal{M}_1 & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}. \tag{20}$$

Proof. As mentioned above, both matrixes $\mathcal{M}_1$ and $\mathcal{M}_2$ are positive definite. Then, we have $\forall \boldsymbol{x}$

$$\boldsymbol{x}^T \mathcal{M}_1 \boldsymbol{x} > 0 \text{ and } \boldsymbol{x}^T \mathcal{M}_2 \boldsymbol{x} > 0.$$

We obtain $\forall \boldsymbol{x}$

$$\boldsymbol{x}^T \mathcal{M}_3 \boldsymbol{x} = \boldsymbol{x}^T \mathcal{M}_1 \boldsymbol{x} + \boldsymbol{x}^T \mathcal{M}_2 \boldsymbol{x} > 0.$$

Therefore, we can conclude that the matrix $\mathcal{M}_3$ is positive definite. On the other hand, if matrix $\mathcal{M}_1$ is positive definite, then we obtain $\forall z = \binom{\boldsymbol{x}}{y}$

$$\begin{aligned} z^T \mathcal{M}_4 z &= \left(\boldsymbol{x}^T \mathcal{M}_1 + y\mathbf{0}, 0\right) \begin{pmatrix} \boldsymbol{x} \\ y \end{pmatrix} \\ &= \boldsymbol{x}^T \mathcal{M}_1 \boldsymbol{x} + y\mathbf{0}\boldsymbol{x} = \boldsymbol{x}^T \mathcal{M}_1 \boldsymbol{x} > 0. \end{aligned}$$

Thus, the matrix $\mathcal{M}_4$ is also positive definite and the result follows. □

Theorem 3.4. *For each cloud player i ($i \in \mathcal{M}$), the set $\mathcal{Q}_i$ is closed and convex, and each disutility function $R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})$ is continuously differentiable in $\boldsymbol{\lambda}_i$. For each fixed tuple $\boldsymbol{\lambda}_{-i}$, the disutility function $R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})$ is convex in $\boldsymbol{\lambda}_i$ over the set $\mathcal{Q}_i$, given that the condition $t \leq \bar{t}$ holds, where*

$$\bar{t} = \frac{m}{((m-2)\boldsymbol{\phi}_{\max} + m\gamma)}, \tag{21}$$

with $\boldsymbol{\phi}_{\max} = \max_{j \in \mathcal{M}}(\boldsymbol{\phi}_j)$.

Proof. To improve the overall readability of this manuscript, the complete proof of this theorem is given in the supplementary material, available online. □

Theorem 3.5. *If the condition (21) holds, then every solution of the variational inequality problem, denoted by $\mathrm{VI}(\mathcal{Q}, \mathbf{F})$, is an equilibrium solution of the game $G = \langle \mathcal{Q}, \mathbf{R} \rangle$, where*

$$\mathbf{F}(\boldsymbol{\lambda}) = (\mathbf{F}_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}))_{i=1}^{m}, \tag{22}$$

with

$$\mathbf{F}_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}) = \nabla_{\boldsymbol{\lambda}_i} R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}). \tag{23}$$

Proof. According to Lemma 4.2 in [37], we know that the above claim holds if two conditions are satisfied. First, for each player i ($i \in \mathcal{M}$), the strategy set $\mathcal{Q}_i$ is closed and convex. Second, for every fixed $\boldsymbol{\lambda}_{-i}$, the disutility function $R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})$ is twice continuously differentiable and convex in $\boldsymbol{\lambda}_i \in \mathcal{Q}_i$. By Theorem 3.2, it is easy to know that both the two conditions are satisfied in the formulated game G. Thus, the result follows. □

Recall that the objective of this section is to study the existence of Nash equilibrium for the formulated game $G = \langle \mathcal{Q}, \mathbf{R} \rangle$ in (18). In the next theorem, we prove that if (21) holds, the existence of such Nash equilibrium is guaranteed.

Theorem 3.6. *There exists a Nash equilibrium solution set for the formulated game $G = \langle \mathcal{Q}, \mathbf{R} \rangle$, given that the condition (21) holds.*

Proof. Based on Theorem 3.3, the proof of this theorem follows if we can show that the formulated variational inequality problem $\mathrm{VI}(\mathcal{Q}, \mathbf{F})$ in Theorem 3.3 possesses a solution set. According to Theorem 4.1 in [37], $\mathrm{VI}(\mathcal{Q}, \mathbf{F})$ admits a solution set if the mapping $\mathbf{F}$ is monotone over $\mathcal{Q}$.

To prove the monotonicity of $\mathbf{F}$, it suffices to show that for any $\boldsymbol{\lambda}$ and $\boldsymbol{s}$ in $\mathcal{Q}$

$$(\boldsymbol{\lambda} - \boldsymbol{s})^T (\mathbf{F}(\boldsymbol{\lambda}) - \mathbf{F}(\boldsymbol{s})) \geq 0,$$

that is

$$\sum_{i=1}^{m} (\boldsymbol{\lambda}_i - \boldsymbol{s}_i)^T (\mathbf{F}_i(\boldsymbol{\lambda}) - \mathbf{F}_i(\boldsymbol{s})) \geq 0. \tag{24}$$

We can observe that if

$$(\boldsymbol{\lambda}_i - \boldsymbol{s}_i)^T (\mathbf{F}_i(\boldsymbol{\lambda}) - \mathbf{F}_i(\boldsymbol{s})) \geq 0,\ \forall i \in \mathcal{M},$$

then inequality (24) holds.

After some algebraic manipulation, we can write the (j,k)th element of $\mathrm{J}_{\boldsymbol{\lambda}_i} \mathbf{F}_i(\boldsymbol{\lambda})$ as

$$\left[\mathrm{J}_{\boldsymbol{\lambda}_i}\mathbf{F}_i(\boldsymbol{\lambda})\right]_{jk} = \frac{\partial^2 R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})}{\partial\lambda_{ij}\partial\lambda_{ik}}$$
$$= \begin{cases} \frac{2\left(1+\alpha_j\right)\left(\left(\mu_j-\beta_j\right)+\left(1+\alpha_j\right)\lambda_{ij}\right)}{\left(\mu_j-\beta_j\right)^3}, & \text{if } i=j=k; \\ \frac{2\left(1+\alpha_j\right)\left(\left(\mu_j-\beta_j\right)+\left(1+\alpha_j\right)\lambda_{ij}\right)}{\left(\mu_j-\beta_j\right)^3} & \\ \quad + \frac{2t\left(t(1-t\gamma)+\lambda_{ij}t^2\right)}{(1-t\gamma)^3}, & \text{if } i\neq j, j=k; \\ \frac{t^2(1-t\gamma)+2\lambda_{ij}t^3}{(1-t\gamma)^3}, & \text{if } i\neq j, i\neq k; \\ 0, & \text{otherwise.} \end{cases}$$

By the derivations in Theorem 3.2, we can conclude that the jacobian matrix $\mathrm{J}_{\boldsymbol{\lambda}_i}\mathbf{F}_i(\boldsymbol{\lambda})$ is positive definite. The result follows. □

3.3 Nash Equilibrium Computation

Once we have established that the Nash equilibrium solution for the formulated game $G = \langle \mathcal{Q}, \mathbf{R}\rangle$ exists, we are interested in obtaining a suitable algorithm to compute one of these equilibria.

Notice that, we can further rewrite the optimization problem (18) as follows:

$$\text{minimize} \quad R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_\Sigma, \gamma),\ \forall i \in \mathcal{M}, \tag{25}$$

where $\boldsymbol{\lambda}_\Sigma$ denotes the aggregated requests of all servers, i.e., $\boldsymbol{\lambda}_\Sigma = \sum_{i=1}^{m}\boldsymbol{\lambda}_i$, and $\gamma = \sum_{i=1}^{m}\sum_{j=1,j\neq i}^{m}\lambda_{ij}$, is the total traffic through the network. From (25), we can observe that the calculation of the disutility function of each individual player only requires the knowledge of the aggregated requests of all servers ($\boldsymbol{\lambda}_\Sigma$) and the total traffic (γ) rather than that the specific individual strategy profile ($\boldsymbol{\lambda}_{-i}$), which can bring about two advantages. On the one hand, it can reduce communication traffic between the virtual communication player and the m players. On the other hand, it can reduce storage for each player to calculate its own strategy.

Since all players are considered to be selfish and try to minimize their own disutilities while ignoring the others. It is natural to consider an iterative algorithm where, at every iteration k, each player i ($\forall i \in \mathcal{M}$) updates its strategy to minimize its own disutility function $R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i})$. However, following [37], it is not difficult to show that their convergence cannot be guaranteed in our case if the players are allowed to simultaneously update their strategies. To overcome this issue, we consider an iterative proximal algorithm, which is based on the best response Algorithm 4.1 [37]. The proposed algorithm is guaranteed to converge to a Nash equilibrium under some additional constraints on the parameters of the algorithm. With reference to [37], we can find a solution to the optimization problem (25) by solving the regularized game in which each of the m players tries to solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & R_i(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_\Sigma, \gamma) + \frac{\tau}{2}\left\|\boldsymbol{\lambda}_i - \bar{\boldsymbol{\lambda}}_i\right\|^2, \\ \text{s.t.} \quad & \boldsymbol{\lambda}_i \in \mathcal{Q}_i,\ \forall i \in \mathcal{M}. \end{aligned} \tag{26}$$

That is to say, when given the external requests, we must find a strategy vector $\boldsymbol{\lambda}^* \in \mathcal{Q}$, such that

$$\boldsymbol{\lambda}_i^* \in \underset{\boldsymbol{\lambda}_i\in\mathcal{Q}_i}{\arg\min}\left\{R_i\left(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_\Sigma^*, \gamma\right) + \frac{\tau}{2}\left\|\boldsymbol{\lambda}_i - \bar{\boldsymbol{\lambda}}_i\right\|^2\right\}, \tag{27}$$

for each player i ($i \in \mathcal{M}$), where τ ($\tau > 0$) is a regularization parameter and can guarantee the convergence of the IPA algorithm if its value is large enough. The idea is formalized in Algorithm 1.

Algorithm 1. Iterative Proximal Algorithm (IPA)

Input: $\epsilon, \boldsymbol{\mu}, \boldsymbol{\lambda}, \mathcal{M}$.
Output: $\boldsymbol{p}_i$.

1: *Initialization*: Randomly choose a feasible strategy vector $\boldsymbol{\lambda}^{(0)}$ ($\boldsymbol{\lambda}^{(0)} \in \mathcal{Q}$). Set $\bar{\boldsymbol{\lambda}} \leftarrow \mathbf{0}$, and $k \leftarrow 0$.
2: **while** ($\left\|\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^{(k-1)}\right\| > \epsilon$) **do**
3: **for** (each player $i \in \mathcal{M}$) **do**
4: Receive $\boldsymbol{\lambda}_\Sigma^{(k)}$ and $\gamma^{(k)}$ from the information player, and compute $\boldsymbol{\lambda}_i^{(k+1)}$ as follows:

$$\boldsymbol{\lambda}_i^{(k+1)} \in \underset{\boldsymbol{\lambda}_i\in\mathcal{Q}_i}{\arg\min}\left\{R_i\left(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_\Sigma^{(k)}, \gamma^{(k)}\right) + \frac{\tau}{2}\left\|\boldsymbol{\lambda}_i - \bar{\boldsymbol{\lambda}}_i\right\|^2\right\}.$$

5: Send the updated strategy to the communication player.
6: **end for**
7: **if** (Nash equilibrium is reached) **then**
8: All of the m players updates their centroids, i.e., $\bar{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}^{(k+1)}$.
9: **end if**
10: Set $k \leftarrow k+1$.
11: **end while**
12: **return** $\boldsymbol{\lambda}^{(k)}$.

At the beginning, each cloud user i ($i \in \mathcal{M}$) sends its server parameters ($\alpha_i, \bar{\mu}_i$) and the external request arrival rate (ϕ_i) to the virtual communication player. Then the communication player computes the regulation parameter (τ) as in Theorem 3.6 according to the aggregated information. After this, the communication player puts the computed regulation parameter into the public information exchange module. Then, at each iteration k, the communication player broadcasts the current aggregated request profile ($\boldsymbol{\lambda}_\Sigma^{(k)}$), and the current communication traffic ($\gamma^{(k)}$). Within iteration k, each player receives the aggregated profile ($\boldsymbol{\lambda}_\Sigma^{(k)}$, $\gamma^{(k)}$) and computes its strategy by solving its own optimization problem in (26), and then sends the newly updated strategy to the communication player. Lastly, as indicated in Algorithm 1 (Steps 7-9), the communication player checks whether the Nash equilibrium has been achieved and if so, it broadcasts a signal to inform all players to update their centroid ($\bar{\boldsymbol{\lambda}}_{\boldsymbol{i}}$). This process continues until the strategies of all players (i.e., the request migration strategies of all servers) are kept fixed. In this paper, we assume that the strategies of all cloud users are unchanged if $\left\|\boldsymbol{\lambda}^{(k)} - \boldsymbol{\lambda}^{(k-1)}\right\| \leq \epsilon$, where $\boldsymbol{\lambda}^{(k)} = (\boldsymbol{\lambda}_i^{(k)})_{i=1}^m$ with $\boldsymbol{\lambda}_i^{(k)} = ((\lambda_{ij})^{(k)})_{j=1}^m$, and ϵ is a relatively small constant.

3.4 Convergence Analysis of $\mathcal{IA}$ Algorithm

In this section, we analyze the convergence of our proposed IPA algorithm. We prove that when the regulation parameter τ is large enough, the IPA algorithm converges to a Nash

equilibrium. Before addressing the convergence problem, we show a property about matrix, which is presented in Theorem 3.5.

Theorem 3.7. *Let* $\mathbf{A} = [a_{ij}]$ *be a real matrix with* m *order, then we have*

$$\|\mathbf{A}\|_2 \leq m\|\mathbf{A}\|_\infty, \tag{28}$$

where $\|\mathbf{A}\|_2$ *and* $\|\mathbf{A}\|_\infty$ *denote the 2-norm and infinite norm of matrix* $\mathbf{A}$*, respectively.*

Proof. Since $\mathbf{A}$ is a real matrix, it is easy to obtain that $\forall \boldsymbol{x}$

$$\boldsymbol{x}^T(\mathbf{A}\mathbf{A}^T)\boldsymbol{x} = (\mathbf{A}^T\boldsymbol{x})^T(\mathbf{A}^T\boldsymbol{x}) \geq 0.$$

That is to say, the matrix $\mathbf{A}^T\mathbf{A}$ is positive semidefinite and we can conclude that $\kappa_{\min}(\mathbf{A}^T\mathbf{A}) \geq 0$ with $\kappa_{\min}(\mathbf{A}^T\mathbf{A})$ denoting the minimal eigenvalue of matrix $\mathbf{A}^T\mathbf{A}$. Let $\kappa_i(\mathbf{A}^T\mathbf{A})$ be the ith eigenvalue of matrix $\mathbf{A}^T\mathbf{A}$, then we have $\kappa_i(\mathbf{A}^T\mathbf{A}) \geq \kappa_{\min}(\mathbf{A}^T\mathbf{A}) \geq 0$. On the other hand, since

$$\sum_{i=1}^m \kappa_i(\mathbf{A}^T\mathbf{A}) = \mathbf{tr}(\mathbf{A}^T\mathbf{A}),$$

where $\mathbf{tr}(\mathbf{A}^T\mathbf{A})$ denotes the summation of all diagonal elements of matrix $\mathbf{A}^T\mathbf{A}$, i.e., $\mathbf{tr}(\mathbf{A}^T\mathbf{A}) = \sqrt{\sum_{i=1}^m \sum_{j=1}^m a_{ij}^2}$, then we have

$$\begin{aligned}\|\mathbf{A}\|_2 &= \sqrt{\kappa_{\max}(\mathbf{A}^T\mathbf{A})} \leq \sqrt{\mathbf{tr}(\mathbf{A}^T\mathbf{A})} \\ &\leq \sqrt{m \cdot \max_{i\in\mathcal{M}} \sum_{j=1}^m \|a_{ij}\|^2} \\ &\leq m \cdot \max_{i\in\mathcal{M}} \sum_{j=1}^m \|a_{ij}\| = m\|\mathbf{A}\|_\infty.\end{aligned}$$

Therefore, we obtain $\|\mathbf{A}\|_2 \leq m\|\mathbf{A}\|_\infty$. This completes the proof and the result follows. □

Theorem 3.8. *There exists a positive constant* τ *such that* $\tau > \tau_0$*, then any sequence* $\{\boldsymbol{\lambda}^{(k)}\}_{k=0}^\infty$ *generated by the* $\mathcal{IA}$ *algorithm converges to a Nash equilibrium, where*

$$\tau_0 = (m-1)m^2\chi_{\max}, \tag{29}$$

with

$$\begin{aligned}\chi_{\max} = &\frac{(1+\alpha_{\max})(\sigma\bar{\mu}_{\min} + 2(1+\alpha_{\max})\phi_{\max})}{(\sigma\bar{\mu}_{\min})^3} \\ &+ \frac{\bar{t}^2(1-\bar{t}\gamma) + 2\phi_{\max}\bar{t}^3}{(1-\bar{t}\gamma)^3},\end{aligned} \tag{30}$$

where $\sigma = \frac{\delta}{1+(1-\delta)\alpha_k}$, $\alpha_{\max} = \max_{k\in\mathcal{M}}(\alpha_k)$, $\bar{\mu}_{\min} = \min_{k\in\mathcal{M}}(\mu_k)$, $\phi_{\max} = \max_{i\in\mathcal{M}}(\phi_i)$, *and*

$$\bar{t} = \frac{m}{((m-2)\phi_{\max} + m\gamma)}. \tag{31}$$

Proof. To improve the overall readability of this manuscript, the complete proof of this theorem is given in the supplementary material, available online. □

4 Performance Evaluation

In this section, we provide some numerical results to validate our theoretical analyses and illustrate the performance of the IPA algorithm. The performance metrics used in our simulation are the multiplied time value (T_R), namely

TABLE 3
System Parameters

System parameters	(Fixed)–[Varied range] (increment)
Total request arrival rate (Φ)	(50)–[50, 500] (25)
Mean communication time (t)	(0.001)–[0.001, 0.06] (0.001, 0.01)
Maximum processing rate ($\bar{\mu}_i$)	[20, 120]
Service deteriorating rate (α_i)	[0, 0.5]
Other parameters (ϵ, δ, m)	(0.01, 0.2, 50)

$$T_R = \prod_{i=1}^m R_i, \tag{32}$$

and specific average response time (R_i), and fairness index ($f(\mathbf{R})$) [36], where

$$f(\mathbf{R}) = \left(\sum_{i=1}^m R_i\right)^2 \Big/ \left(m\sum_{i=1}^m R_i^2\right), \tag{33}$$

is used to quantify the fairness of load balancing schemes. If all servers have the same expected response time, then $f = 1$ and the system is fair. Notice that, different from the central view, our method is developed in a distributed and non-cooperative environment.

In the following simulation results, we consider a scenario that the maximum total external request arrival rate (Φ) can be 500 and the maximum mean request communication time can be 0.06. As shown in Table 3, the total external request arrival rate (Φ) is varied from 50 to 500 with increment 25. The mean request communication time (t) is varied from 0.001 to 0.01 with increment 0.001, and from 0.01 to 0.06 with increment 0.01. The maximum processing rate of a server ($\bar{\mu}_i$) and its corresponding deteriorating rate (α_i) are randomly chosen from 20 to 120 and 0 to 0.5, respectively. The number of servers (m) in the cloud provider is set as a constant 50. In our configuration, δ is set as 0.2, that is to say, the aggregated requests at a server cannot exceed 80 percent expected processing capability of the server, and the accuracy control parameter (ϵ) is set as 0.01. Of course, other parameter values can also be configured.

Fig. 4 shows an instance for the average response times of different servers versus the number of iterations of the proposed IPA algorithm. Specifically, Fig. 4 illustrates the average response time trends of 6 randomly selected servers (servers 7, 11, 17, 24, 35, and 50) during the iteration process (the iteration process of the while loop) of our proposed IPA algorithm. In this scenario, the total number of servers is 50. From Fig. 4, we can observe that the average response times of some servers (servers 11, 17, 24, 35, and 50) tend to decrease with the increase of the iteration number while some servers (server 7) tend to increase. However, all the average response times reach a relatively stable state, which verifies the convergency result shown in Theorem 3.8. That is, the average response times of all servers are kept unchanged after several iterations, i.e., reach a Nash equilibrium solution after several iterations. In addition, it can also be seen that the developed algorithm converges to a Nash equilibrium very quickly. Specifically, the average response times of all servers have already achieved a relatively stable

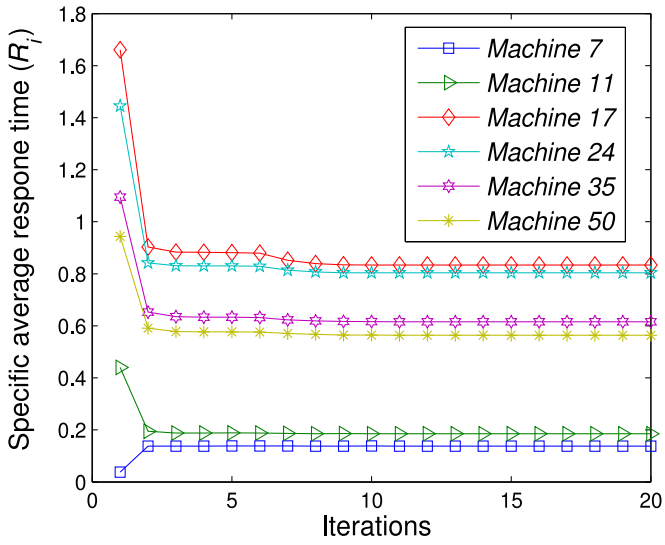


Fig. 4. Convergency process of specific response time.

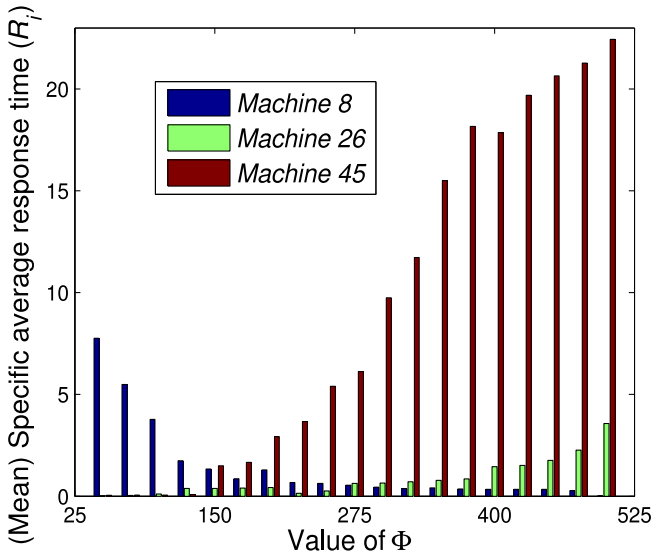


Fig. 6. Impacts of Φ on specific response time.

state after around 8 iterations, which shows the high efficiency of our developed algorithm.

In Fig. 5, we compare the multiplied time values (calculated according to Eq. (32)) with the situations before and after IPA algorithm, i.e., compare the results without request migration and those with request migration configured by our IPA algorithm. Specifically, Fig. 5 shows the multiplied time values with the increase of total external request arrival rate. We conduct 300 times with initial strategies randomly chosen from the feasible strategy set, and present the mean value in Fig. 5. The initial multiplied time value (IT_R) corresponds to the result of a feasible strategy profile randomly generated in the initialization stage, i.e., the profile without request migration, while the result (T_R) corresponds to the value obtained by adopting our request migration scheme. Obviously, our IPA algorithm can significantly optimize the multiplied value generated by initial strategy, which shows that our proposed algorithm is effective for load balance under distributed environment. We can also observe that the trend of multiplied values tend to increase with increase of total external request arrival rate. The reason behind lies in that with the increase of total external requests, the expected response times of some servers significantly increase. To demonstrate this phenomenon, we further investigate the specific response times of different servers. The results are presented in Fig. 6.

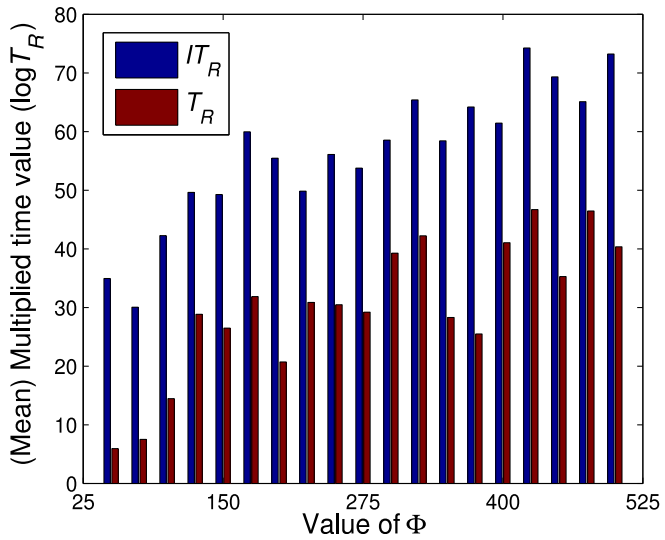


Fig. 5. Impacts of Φ on multiplied time value.

In Fig. 6, we plot the average response time shapes of some servers for the developed IPA algorithm with the increase of total external request arrival rate. Specifically, in Fig. 6, we randomly select 3 servers (servers 8, 26, and 45) and show the expected response time trends. It can be seen that the average response times of some servers tend to increase with the increase of total external request arrival rate (servers 26 and 45), while some of them tend to decrease (server 8). The reason behind lies in that some slow servers tend to migrate their arrived requests to other servers to decrease their own expected response times. Otherwise, their expected response times will significantly increase even with a small load increase. On the other hand, some fast servers tend to process the arrived requests by themselves. Hence, the expected response times of some servers tend to increase while some servers tend to decrease. Fig. 7 presents the fairness results. The initial fairness index (If) corresponds to the initial strategy, i.e., the strategy without request migration, and the other corresponds to the strategy obtained by our IPA algorithm. We can observe that at first, both results are small and the result obtained by our algorithm is even worse. With the increase of total request arrival rate, both results tend to increase. However, the result obtained by IPA algorithm is more better. The reason behind lies in that when the total request is small, there are no aggregated requests on many servers, i.e., many servers run with very little requests compared to their whole processing capacities. However, with the increase of total request arrival rate, our method can find a better balancing strategy and reaches around 0.6-0.7 fairness index in our non-cooperative environment.

Figs. 8 and 9 present the impacts of mean request communication time on multiplied time value and the corresponding specific response times of some servers. Specifically, Fig. 8 illustrates the multiplied time value with the

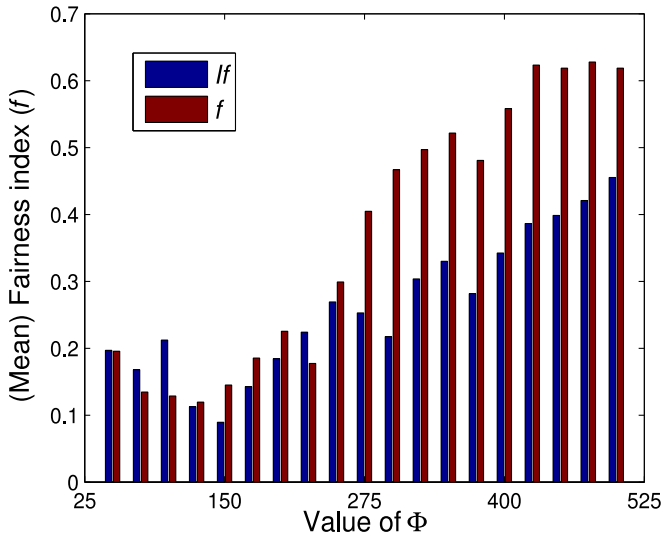


Fig. 7. Impacts of Φ on fairness index.

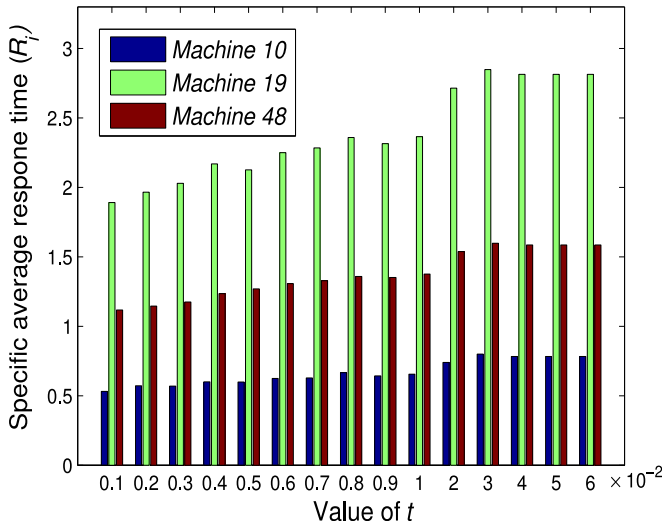


Fig. 9. Impacts of t on specific response time.

increment of mean communication time. We can observe that the multiplied time value increases with increase of mean request communication time (from 0.001 to 0.03). However, it keeps unchanged if the mean communication time of a request is large enough (from 0.04 to 0.06). Fig. 9 shows the corresponding specific average response times. We randomly select 3 servers (servers 10, 19, and 48). It can be seen that the average response times of servers tend to increase with the increase of mean request communication time (from 0.001 to 0.03) at first. Then, they are kept unchanged when the mean communication time is large enough (from 0.04 to 0.06). The reason behind lies in that at the beginning, even though the mean request communication time increases, some servers still prefer to migrate some of their requests to other servers to reduce their own expected response times. However, when the mean request communication time (i.e., the cost to migrate a request) is large enough, they will prefer to complete all the aggregated requests on their own. This also partly verifies the multiplied time shape shown in Fig. 8.

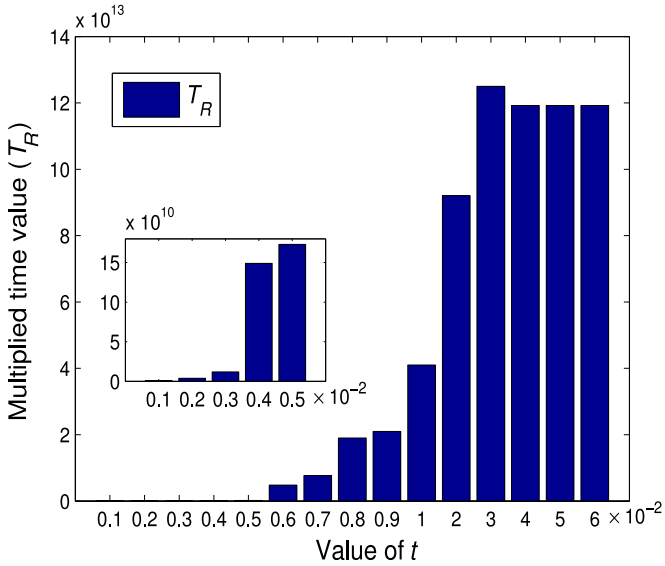


Fig. 8. Impacts of t on multiplied time value.

5 Conclusions

An increasing number of applications migrated to cloud centers, load balancing has become one of the most important factors for service quality. However, most of the existing scheduling algorithms in clouds ignore the server availability, which can lead to load imbalance and a great waste of computing resources. To remedy this problem to certain extent, we propose a non-cooperative game based load balancing scheme, which involves load-dependent server availability.

In this paper, we focus on request migration strategies of multiple servers for load balance in cloud. We consider the problem from a game theoretic perspective and formulate it into a non-cooperative game among the multiple servers, in which each server is informed with incomplete information of other servers. For each server, we define its average response time as a disutility function and try to minimize its value. We also take into account server availability, which impacts the processing rate of a server and thus its disutility. We solve the problem by employing variational inequality theory and prove that there exists a Nash equilibrium solution set for the formulated game. Then, we propose an iterative proximal algorithm to compute a Nash equilibrium solution. The convergence of the IPA algorithm is also analyzed and we find that it converges to a Nash equilibrium if several conditions are satisfied. Finally, we conduct some numerical calculations to verify our theoretical analysis. The experimental results show that our proposed IPA algorithm converges to a Nash equilibrium very quickly and significantly decreases the disutilities of all servers by configuring a proper request migration strategy.

As part of future directions, we will extend IPA to request migration across clouds, in which the communication cost can be large. Another direction is to study the integration of our IPA with some pre-scheduling algorithm.

Acknowledgments

We are very grateful to the associate editor and anonymous reviewers for their comments and suggestions which have significantly improved the quality of the manuscript. The research was partially funded by the National Key R&D

Program of China (Grant No. 2016YFB0201402), the National Natural Science Foundation of China (Grant Nos. 61702170, 61602350, 61602170, 61402400, 61370098, 61672219), the Key Program of National Natural Science Foundation of China (Grant No. 61432005), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the National High-tech R&D Program of China (2015AA015305), and the Chinese Postdoctoral Science Foundation (Grant Nos. 2016M602409, 2016M602410)

REFERENCES

[1] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 45–58, Jan. 2014.

[2] J. Mei, K. Li, and K. Li, "Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing," *IEEE Trans. Sustainable Comput.*, vol. 2, no. 1, pp. 17–29, Jan.–Mar. 2017.

[3] P. D. Kaur and I. Chana, "A resource elasticity framework for QoS-aware execution of cloud applications," *Future Generation Comput. Syst.*, vol. 37, pp. 14–25, 2014.

[4] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[5] S. Penmatsa and A. T. Chronopoulos, "Game-theoretic static load balancing for distributed systems," *J. Parallel Distrib. Comput.*, vol. 71, no. 4, pp. 537–555, 2011.

[6] C.-W. Ang and C.-K. Tham, "Analysis and optimization of service availability in a HA cluster with load-dependent machine availability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 9, pp. 1307–1319, Sep. 2007.

[7] H. Khazaei, J. Misic, and V. Misic, "A fine-grained performance model of cloud computing centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 11, pp. 2138–2147, Nov. 2013.

[8] A. Singh, D. Juneja, and M. Malhotra, "Autonomous agent based load balancing algorithm in cloud computing," *Procedia Comput. Sci.*, vol. 45, pp. 832–841, 2015.

[9] A. Paya and D. C. Marinescu, "Energy-aware load balancing and application scaling for the cloud ecosystem," *IEEE Trans. Cloud Comput.*, vol. 5, no. 1, pp. 15–27, Jan.–Mar. 2017.

[10] Y. Feng, B. Li, and B. Li, "Price competition in an oligopoly market with multiple IaaS cloud providers," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 59–73, Jan. 2014.

[11] A. Kassab, J.-M. Nicod, L. Philippe, and V. Rehn-Sonigo, "Scheduling independent tasks in parallel under power constraints," in *Proc. 46th Int. Conf. Parallel Process.*, 2017, pp. 543–552.

[12] J. Zhao, K. Yang, X. Wei, Y. Ding, L. Hu, and G. Xu, "A heuristic clustering-based task deployment approach for load balancing using bayes theorem in cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 305–316, Feb. 2016.

[13] M. J. Siavoshani, S. P. Shariatpanahi, H. Ghasemi, and A. Pourmiri, "On communication cost versus load balancing in content delivery networks," in *Proc. IEEE Symp. Comput. Commun.*, 2017, pp. 651–656.

[14] P. T. Thant, C. Powell, M. Schlueter, and M. Munetomo, "A level-wise load balanced scientific workflow execution optimization using NSGA-II," in *Proc. 17th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2017, pp. 882–889.

[15] H. Shen, "RIAL: Resource intensity aware load balancing in clouds," *IEEE Trans. Cloud Comput.*, 2017.

[16] G. Scutari, D. P. Palomar, F. Facchinei, and J.-S. Pang, "Convex optimization, game theory, and variational inequality theory," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 35–49, May 2010.

[17] J.-P. Aubin, *Mathematical Methods of Game and Economic Theory*. North Chelmsford, MA, USA: Courier Corporation, 2007.

[18] S. S. Aote and M. Kharat, "A game-theoretic model for dynamic load balancing in distributed systems," in *Proc. Int. Conf. Advances Comput. Commun. Control*, 2009, pp. 235–238.

[19] N. Li and J. R. Marden, "Designing games for distributed optimization," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 2, pp. 230–242, Apr. 2013.

[20] N. S. Rao, S. W. Poole, F. He, J. Zhuang, C. Y. Ma, and D. K. Yau, "Cloud computing infrastructure robustness: A game theory approach," in *Proc. Int. Conf. Comput. Netw. Commun.*, 2012, pp. 34–38.

[21] X. Xu and H. Yu, "A game theory approach to fair and efficient resource allocation in cloud computing," *Math. Problems Eng.*, vol. 2014, 2014, Art. no. 915878.

[22] J. Künsemöller and H. Karl, "A game-theoretical approach to the benefits of cloud computing," in *Economics of Grids, Clouds, Systems, and Services*. Berlin, Germany: Springer, 2012, pp. 148–160.

[23] G. Scutari, F. Facchinei, J.-S. Pang, and D. P. Palomar, "Real and complex monotone communication games," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 4197–4231, Jul. 2014.

[24] K. Li, C. Liu, K. Li, and A. Y. Zomaya, "A framework of price bidding configurations for resource usage in cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2168–2181, Aug. 2016.

[25] A.-H. Mohsenian-Rad, V. W. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia, "Autonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid," *IEEE Trans. Smart Grid*, vol. 1, no. 3, pp. 320–331, Dec. 2010.

[26] I. Atzeni, L. G. Ordóñez, G. Scutari, D. P. Palomar, and J. R. Fonollosa, "Noncooperative and cooperative optimization of distributed energy generation and storage in the demand-side of the smart grid," *IEEE Trans. Signal Process.*, vol. 61, no. 10, pp. 2454–2472, May 2013.

[27] C. Liu, K. Li, C. Xu, and K. Li, "Strategy configurations of multiple users competition for cloud service reservation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 508–520, Feb. 2016.

[28] E. G. Larsson, E. Jorswieck, J. Lindblom, and R. Mochaourab, "Game theory and the flat-fading gaussian interference channel," *IEEE Signal Process. Mag.*, vol. 26, no. 5, pp. 18–27, Sep. 2009.

[29] X. Qin and T. Xie, "An availability-aware task scheduling strategy for heterogeneous systems," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 188–199, Feb. 2008.

[30] B. Vahedi-Nouri, P. Fattahi, M. Rohaninejad, and R. Tavakkoli-Moghaddam, "Minimizing the total completion time on a single machine with the learning effect and multiple availability constraints," *Appl. Math. Modelling*, vol. 37, no. 5, pp. 3126–3137, 2013.

[31] C. Zhao, M. Ji, and H. Tang, "Parallel-machine scheduling with an availability constraint," *Comput. Ind. Eng.*, vol. 61, no. 3, pp. 778–781, 2011.

[32] I. Kacem, "Effective algorithms for scheduling problems under non-availability constraints," in *Proc. 2nd Int. Conf. Syst. Comput. Sci.*, 2013, pp. 162–169.

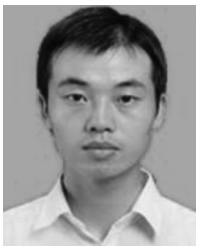
[33] L.-W. Liao and G.-J. Sheen, "Parallel machine scheduling with machine availability and eligibility constraints," *Eur. J. Oper. Res.*, vol. 184, no. 2, pp. 458–467, 2008.

[34] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Depend. Secure Comput.*, vol. 7, no. 4, pp. 337–350, Oct.–Dec. 2010.

[35] J. Tian, S. Rudraraju, and Z. Li, "Evaluating web software reliability based on workload and failure data extracted from server logs," *IEEE Trans. Softw. Eng.*, vol. 30, no. 11, pp. 754–769, Nov. 2004.

[36] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Hoboken, NJ, USA: Wiley, 1990.

[37] G. Scutari, D. P. Palomar, F. Facchinei, and J.-S. Pang, "Monotone games for cognitive radio systems," in *Distributed Decision Making and Control*. Berlin, Germany: Springer, 2012, pp. 83–112.



Chubo Liu received the BS and PhD degrees in computer science and technology from Hunan University, China, in 2011 and 2016, respectively. His research interests include mainly in modeling and scheduling of distributed computing systems, approximation and randomized algorithms, game theory, grid, and cloud computing. He has published over 5 papers in journals such as the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Cloud Computing*, the *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, the *Future Generation Computer Systems*, and the *Theoretical Computer Science*.

Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently the dean and a full professor of computer science and technology with Hunan University and director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid, and cloud computing. He has published more than 150 research papers in international conferences and journals such as the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Signal Processing*, the *Journal of Parallel and Distributed Systems*, ICPP, and CCGrid. He serves on the editorial board of the *IEEE Transactions on Computers*. He is an outstanding member of CCF. He is a senior member of the IEEE.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems. He has published more than 480 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.