Bargaining Game-Based Scheduling for Performance Guarantees in Cloud Computing

CHUBO LIU, KENLI LI, and ZHUO TANG, Hunan University
KEQIN LI, State University of New York

In this article, we focus on request scheduling with performance guarantees of all users in cloud computing. Each cloud user submits requests with average response time requirement, and the cloud provider tries to find a scheduling scheme, i.e., allocating user requests to limited servers, such that the average response times of all cloud users can be guaranteed. We formulate the considered scenario into a cooperative game among multiple users and try to find a Nash bargaining solution (NBS), which can simultaneously satisfy all users' performance demands. We first prove the existence of NBS and then analyze its computation. Specifically, for the situation when all allocating substreams are strictly positive, we propose a computational algorithm ($\mathcal{CA}$), which can find the NBS very efficiently. For the more general case, we propose an iterative algorithm ($\mathcal{IA}$), which is based on duality theory. The convergence of our proposed $\mathcal{IA}$ algorithm is also analyzed. Finally, we conduct some numerical calculations. The experimental results show that our $\mathcal{IA}$ algorithm can find an appropriate scheduling strategy and converges to a stable state very quickly.

CCS Concepts: • **Computing methodologies** → *Modeling methodologies*;

Additional Key Words and Phrases: Cloud computing, cooperative game, Nash bargaining solution, performance guarantees

ACM Reference format:
Chubo Liu, Kenli Li, Zhuo Tang, and Keqin Li. 2018. Bargaining Game-Based Scheduling for Performance Guarantees in Cloud Computing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 3, 1, Article 1 (February 2018), 25 pages.
https://doi.org/10.1145/3141233

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, 61772182, 61572176, and L1624040), 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), the Key Technology Research and Development Programs of Guangdong Province (2015B010108006), the International S&T Cooperation Program of China (2015DFA11240), and the Chinese Postdoctoral Science Foundation (Grant Nos. 2016M602409 and 2016M602410).
Authors' addresses: C. Liu, K. Li, and Z. Tang, College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Changsha, Hunan 410082, China; K. Li, Department of Computer Science, State University of New York, New Paltz, New York 12561; emails: liuchubo@hnu.edu.cn, lkl@hnu.edu.cn, ztang@hnu.edu.cn, lik@newpaltz.edu.

https://doi.org/10.1145/3141233

1 INTRODUCTION

1.1 Motivation

Cloud computing is a large-scale distributed computing paradigm in which a pool of computing resources is available to users via the Internet (Chaisiri et al. 2012). With the scalability, cost-effectiveness, and many other advantages of cloud computing, more and more user applications are moved from local to cloud centers (Khazaei et al. 2013). However, the increasing resource demands with different computation requirements from such applications also raise some issues such as server underutilization and poor service quality (Kang et al. 2014b). Hence, it is important for a cloud provider to configure an appropriate scheduling scheme, which significantly impacts the aggregated performance and resource utilization of a cloud center.

Cloud-based applications depend even more heavily on scheduling than traditional enterprise applications (Cao et al. 2014). For end users, scheduling will be seriously considered when they select a cloud computing provider. For a cloud provider, an appropriate scheduling scheme becomes particularly important (Mei et al. 2015). The reason behind this lies in that a scheduling scheme is directly related to service quality and profit of a cloud provider. Specifically, a proper scheduling scheme helps in making use of the available resources most favorably. This implies that a cloud provider can use less computing resources to provide quality services and thus decrease its cost. On the other hand, an appropriate scheduling scheme improves the aggregated performance (e.g., task response time). This will satisfy users and appeal to more potential users in the market to use cloud servics.

Many works have been done on request scheduling for various considerations in the literature (Singh et al. 2015; Rana et al. 2014; Mashaly and Kühn 2012; Renda et al. 2012). However, few works can be found for scheduling schemes which simultaneously guarantee the performance requirements of all cloud users. In this work, although we only consider an average response time metric, we try to find a scheduling scheme which satisfies the average response time requirements of all cloud users at the same time.

1.2 Our Contributions

We focus on request scheduling with performance guarantees of all users in cloud computing. Each cloud user submits requests with an average response time requirement. The cloud provider tries to find a scheduling scheme, i.e., allocating requests to multiple servers, such that the average response time requirements of all cloud users can be guaranteed. In our work, we formulate the considered scenario into a cooperative game among multiple cloud users. We study the cooperative relationships and propose algorithms to configure an appropriate request allocation strategy for each of the users.

In summary, the main contributions of this work can be listed as follows:

- We formulate the scheduling problem into a cooperative game and try to find a Nash bargaining solution (NBS) which can satisfy the average response time requirements of all cloud users at the same time.
- For the situation in which each allocating substream is strictly positive, we propose a computational algorithm ($\mathcal{CA}$) which can efficiently find the NBS.
- For the general case, after some observations and analyses, we propose an iterative algorithm ($\mathcal{IA}$) which is based on duality theory.

We also perform extensive experiments. The results show that our proposed methods are feasible and effective.

The rest of the article is organized as follows. Section 2 presents the relevant works. Section 3 describes the models of the system and presents the problem to be solved. Section 4 presents the bargaining game-based method. Many analyses are also presented in this section. Section 5 is developed to verify our theoretical analysis and show the feasibility of our proposed algorithm. We conclude the article with future work in Section 6.

2 RELATED WORK

Many works have been done on task scheduling in the literature (Cao et al. 2014; Penmatsa and Chronopoulos 2011; Dong et al. 2012; Singh et al. 2015; Xu et al. 2012; Mc Evoy and Schulze 2011; Kang et al. 2014a). In Cao et al. (2014), Cao et al. studied the tradeoff between the aggregated performance and energy constraint in cloud. Specifically, they studied scheduling schemes with the situation when energy consumption is limited. Dong et al. (2012) presented a dynamic and adaptive scheduling algorithm, which is based on a distributed architecture, to address the prohibitively inefficient problem in large-scale parallel file systems. In Singh et al. (2015), the authors proposed an autonomous agent-based load balancing algorithm, which is dynamic to cater the requirements in cloud computing environment. More relevantly, Penmatsa and Chronopoulos (2011) proposed a cooperative game-based CCOOP algorithm, in which they tried to minimize the average response times of servers. More works can be found in Xu et al. (2012), Mc Evoy and Schulze (2011), and Kang et al. (2014a). Different from their considerations, we focus on service quality guided scheduling. We try to find a scheduling scheme such that the average response time requirements of all cloud users can be guaranteed.

When considering a scheduling which guarantees the performance requirements of all cloud users, the problem becomes more complex. Few works can be found for this in the literature. In Subrata et al. (2008), Subrata et al. used cooperative game to allocate tasks among multiple cloud providers. In this work, we also use the cooperative game method. However, the models and objectives are entirely different. The analyzing methods and solving strategies are also different.

Game theory is a field of applied mathematics that describes and analyzes scenarios with interactive decisions (Scutari et al. 2010; Osborne and Rubinstein 1994; Aubin 2007). It is a formal study of conflicts and cooperations among multiple competitive users (Aote and Kharat 2009) and a powerful tool for the design and control of multiagent systems (Li and Marden 2011). There has been a growing interest in adopting cooperative and non-cooperative game approaches to modeling many problems (Rao et al. 2012; Xu and Yu 2014; Künsemöller and Karl 2012). In Rao et al. (2012), the authors presented a game-theoretic approach for the provisioning of the infrastructure under uniform cost models. Xu and Yu (2014) proposed a game-theoretic resource allocation algorithm which considers resource utilization. In Gao et al. (2012), the authors employed indirect reciprocity game to study the incentives for cooperative communications among multiple users. Musku et al. (2010) also studied communication problems. More specifically, they used non-cooperative game to analyze the joint transmission rate and power control for the uplink of a single cell CDMA system. In their work, the authors analyzed the existence of the Nash equilibrium solution and proposed an algorithm to compute an equilibrium solution. In our previous work (Liu et al. 2016), we used non-cooperative game theory to analyze the utility optimization strategy for each of the cloud users. Specifically, we try to obtain a Nash equilibrium to optimize all users' utilities. For more works on game theory, the reader is referred to Grossi and Turrini (2010), Blum et al. (2015), Thompson and Leyton-Brown (2013), Kilcioglu and Rao (2016), Li et al. (2014), and Fiat et al. (2013).

3 SYSTEM MODEL AND PROBLEM FORMULATION

In this article, we are concerned with a market with a cloud provider and n cloud users, who are competing for using the computing resources provided by the cloud provider. We denote the set

Table 1. Notations

Notation	Description
n	Number of cloud users
m	Number of servers in the cloud center
$\mathcal{N}$	Set of the n cloud users
$\mathcal{M}$	Set of the m servers in the cloud center
μ_j	The processing rate of a core of server j
c_j	The number of cores of server j
λ_{ij}	Allocating substream of requests from user i to server j
$\boldsymbol{\lambda}_i$	Allocating strategy for requests of cloud user i
$\boldsymbol{\lambda}^j$	Allocating strategy for requests aggregated on server j
$\boldsymbol{\lambda}$	Allocating strategy for requests of all cloud users
$\boldsymbol{Q}$	Request allocation strategy set of all cloud users
Λ_i	Arrival rate of requests of cloud user i
χ_j	Aggregated requests on server j
T_j	Average response time of server j
R_i	Average response time of cloud user i
$\boldsymbol{R}$	Average response times of all cloud users
$R_{i,\max}$	Performance requirement of cloud user i
$\boldsymbol{R}_{\max}$	Performance requirements of all cloud users
$\mathcal{U}$	Set of achievable utility vectors of all cloud users
$\mathcal{A}$	Set of achievable utility vectors satisfying requirements
$\mathcal{P}$	Pair set of utility and performance requirements
S	Mapping from $\mathcal{P}$ to $\Re^n$

of users as $\mathcal{N} = \{1, \ldots, n\}$. Each cloud user submits requests with an average response time requirement. The arrival requests from cloud user i ($i \in \mathcal{N}$) are assumed to follow a Poisson process. The cloud provider consists of m heterogeneous servers. Denote the server set as $\mathcal{M} = \{1, \ldots, m\}$. Each server j ($j \in \mathcal{M}$) has c_j identical cores and is modeled as an M/M/c queuing system, which is commonly used in scheduling literature (Cao et al. 2014; Li 2013). The processing capacity of a core of server j ($j \in \mathcal{M}$) is presented by its service rate μ_j. The cloud provider tries to find a request scheduling scheme such that the average response time requirements of all cloud users can be guaranteed.

We summarize all the notations throughout this article in Table 1.

3.1 Request Allocation Strategy Model

As mentioned above, the requests from each of the cloud users are assumed to follow a Poisson process. Let λ_{ij} be the substream of requests from cloud user i ($i \in \mathcal{N}$) allocated to server j ($j \in \mathcal{M}$). Then, user i's ($i \in \mathcal{N}$) request allocation profile is formulated as

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

and it is subject to the constraint $\sum_{j=1}^{m} \lambda_{ij} = \Lambda_i$, where Λ_i denotes user i's request arrival rate. We can further obtain the request allocation strategy of all cloud users as

$$\boldsymbol{\lambda} = (\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_n). \tag{2}$$

Notice that the aggregated requests on a server cannot exceed its processing capacity, i.e., $\sum_{i=1}^{n} \lambda_{ij} < c_j \mu_j$, for all $j \in \mathcal{M}$. In our work, we assume that $\sum_{i=1}^{n} \lambda_{ij} \leq c_j \hat{\mu}_j$, where $\hat{\mu}_j = \mu_j - \epsilon$ with

ϵ denoting a very small positive constant. Therefore, the request allocation strategy set of all cloud users, $\boldsymbol{Q}$, is defined by the following constraints:

$$\begin{cases} \sum_{j=1}^{m} \lambda_{ij} = \Lambda_i, \forall i \in \mathcal{N}, \\ \sum_{i=1}^{n} \lambda_{ij} \le c_j \hat{\mu}_j, \forall j \in \mathcal{M}, \\ \lambda_{ij} \ge 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}. \end{cases} \tag{3}$$

3.2 Cloud Service Model

As mentioned in the beginning, each server j ($j \in \mathcal{M}$) has c_j identical cores and is modeled as an M/M/c queuing system. The processing capacity of each core of server j ($j \in \mathcal{M}$) is presented by its service rate μ_j. The requests from cloud user i ($i \in \mathcal{N}$) are assumed to follow a Poisson process with mean arrival rate Λ_i.

Let p_{jk} be the probability that there are k service requests (waiting or being processed) and $\rho_j = \sum_{i=1}^{n} \lambda_{ij}/(c_j \mu_j)$ be the corresponding service utilization in the M/M/c queuing system on server j. With reference to Cao et al. (2013), we obtain

$$p_{jk} = \begin{cases} \frac{1}{k!}(c_j \rho_j)^k p_{j0}, & k < c_j; \\ \frac{c_j^{c_j} \rho_j^k}{c_j!} p_{j0}, & k \ge c_j; \end{cases} \tag{4}$$

where

$$p_{j0} = \left\{ \sum_{l=0}^{c_j - 1} \frac{1}{l!}(c_j \rho_j)^l + \frac{1}{c_j!} \cdot \frac{(c_j \rho_j)^{c_j}}{1 - \rho_j} \right\}^{-1}. \tag{5}$$

The average number of service requests (in waiting or in execution) on server j is

$$\bar{N}_j = \sum_{k=0}^{\infty} k p_{jk} = \frac{p_{jc_j}}{1 - \rho_i} = c_j \rho_j + \frac{\rho_j}{1 - \rho_j} P_j, \tag{6}$$

where P_j represents the probability that the incoming requests on server j ($j \in \mathcal{M}$) need to wait in queue.

Applying Little's result, we get the average response time of server j as

$$T_j = \frac{\bar{N}_j}{\chi_j} = \frac{1}{\chi_j} \left(c_j \rho_j + \frac{\rho_j}{1 - \rho_j} P_j \right), \tag{7}$$

where χ_j denotes the aggregated requests on server j, i.e., $\chi_j = \sum_{i=1}^{n} \lambda_{ij}$. We assume that each of the servers will likely keep busy, because if not, we can shut down some servers to reduce cost. Hence, P_j is equal to 1, and we obtain

$$T_j = \frac{\bar{N}_j}{\chi_j} = \frac{1}{\chi_j} \left(c_j \rho_j + \frac{\rho_j}{1 - \rho_j} \right) = \frac{1}{\mu_j} + \frac{1}{c_j \mu_j - \chi_j}. \tag{8}$$

3.3 Architecture Model

As shown in Figure 1, each cloud user i ($i \in \mathcal{N}$) is equipped with a request arrival rate (Λ_i) and the average response time requirement ($R_{i,\max}$), i.e., the maximum average response time the user can tolerate. All requests enter a queue to be processed by the cloud center with first come first service (FCFS) pattern. The cloud provider consists of m heterogeneous servers. Each server j ($j \in \mathcal{M}$) has c_j identical cores with total processing capacity $c_j \mu_j$, where μ_j is the service rate of a core of server j. When multiple users try to accomplish their requests in cloud, they submit their requests with average response time requirements. The cloud provider collects the request arrival rates

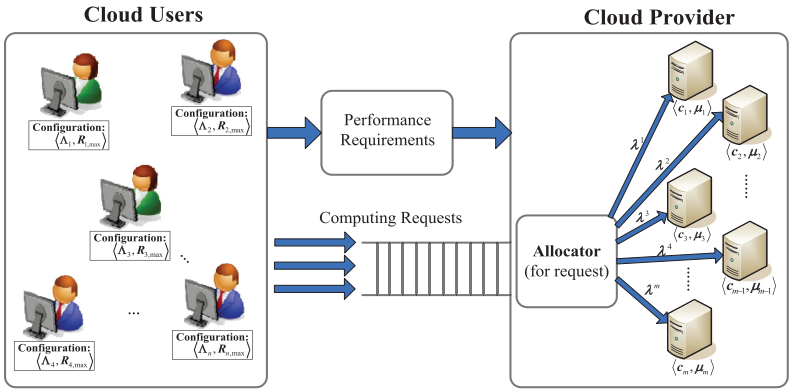


Fig. 1. Architecture model.

and requirement information of all users, and then makes appropriate scheduling decisions ($\boldsymbol{\lambda}$) for each of the users. As presented before, the scheduling strategy $\boldsymbol{\lambda}$ can be expressed as $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_i)_{i\in\mathcal{N}}$, where $\boldsymbol{\lambda}_i = (\lambda_{ij})_{j\in\mathcal{M}}$ with λ_{ij} denoting the substream of requests from cloud user i allocated to server j. Let $\boldsymbol{\lambda}^j$ ($j \in \mathcal{M}$) denote the scheduling strategy vector of all cloud users on server j, i.e., $\boldsymbol{\lambda}^j = (\lambda_{ij})_{i\in\mathcal{N}}$, then the scheduling strategy can also be expressed as $\boldsymbol{\lambda} = (\boldsymbol{\lambda}^j)_{j\in\mathcal{M}}$ (see Figure 1). The cloud provider tries to find a scheduling strategy ($\boldsymbol{\lambda}$) such that the average response time requirements of all cloud users can be guaranteed.

3.4 Problem Formulation

In Equation (8), we have presented the calculation of the average response time of a single server. Therefore, for a given request allocation strategy $\boldsymbol{\lambda}$ of all cloud users, the average response time of cloud user i ($i \in \mathcal{N}$) can be expressed as

$$\begin{aligned} R_i(\boldsymbol{\lambda}) &= \frac{1}{\Lambda_i}\sum_{j=1}^{m}\lambda_{ij}T_j(\boldsymbol{\lambda}) \\ &= \frac{1}{\Lambda_i}\sum_{j=1}^{m}\left(\frac{\lambda_{ij}}{\mu_j} + \frac{\lambda_{ij}}{c_j\mu_j - \chi_j(\boldsymbol{\lambda})}\right), \end{aligned} \tag{9}$$

where Λ_i is the request arrival rate of user i, and $\chi_j(\boldsymbol{\lambda})$ denotes the aggregated requests on server j under request allocation strategy $\boldsymbol{\lambda}$, i.e., $\chi_j(\boldsymbol{\lambda}) = \sum_{i=1}^{n}\lambda_{ij}$.

We consider the scenario where the average response time requirements of all cloud users are guaranteed. Specifically, the cloud provider tries to find a request allocation strategy such that the average response time of each cloud user i ($i \in \mathcal{N}$) is less than his/her demand $R_{i,\max}$, i.e., find an allocation strategy $\boldsymbol{\lambda}$ satisfying the following condition:

$$R_i(\boldsymbol{\lambda}) \le R_{i,\max}, \forall i \in \mathcal{N}, \boldsymbol{\lambda} \in Q. \tag{10}$$

Remark 3.1. A scheduling strategy ($\boldsymbol{\lambda}$) satisfying Equation (10) guarantees that the average response times of all cloud users are simultaneously less than their corresponding performance demands.

4 BARGAINING GAME-BASED METHOD

To obtain a scheduling strategy such that the performance requirements of all cloud users are guaranteed, we try to obtain a NBS which can satisfy the performance requirements of all users

by appropriately modeling. We first prove the existence of NBS and then analyze its computing method. For the situation when all allocating substreams are strictly positive, we propose a $\mathcal{CA}$, which can find the NBS very efficiently. For the more general case, we propose an $\mathcal{IA}$, which is based on duality theory.

4.1 Game Formulation

Game theory studies the problems in which players try to maximize their utilities or minimize their disutilities. As described in Penmatsa and Chronopoulos (2011), a cooperative game consists of a set of players, a set of utility functions, and a set of strategies. In this article, each cloud user is regarded as a player, i.e., the set of players is the n cloud users. The utility function of player i ($i \in \mathcal{N}$) is his/her average response time, i.e., R_i, and the joint strategy set of all players is given by $\mathcal{Q}$, which is defined by the constraints in Equation (3).

The above formulated game can be formally defined by a tuple $\mathcal{G} = \langle \mathcal{Q}, \boldsymbol{R} \rangle$, where $\boldsymbol{R} = (R_1, \ldots, R_n)$. The optimization goal is to determine a request allocation strategy which guarantees the average response time requirements of all cloud users. Let $\mathcal{U} \subset \Re^n$ be the set of achievable utility vectors and $\boldsymbol{R}_{\max}$ be the corresponding average response time requirements of all cloud users, i.e., $\boldsymbol{R}_{\max} = (R_{i,\max})_{i \in \mathcal{N}}$. Then, the pair $(\mathcal{U}, \boldsymbol{R}_{\max})$ is called an n-player bargaining problem. Before addressing the bargaining problem, we first define the notation of Pareto optimality in the context of multiple objectives within the feasible set $\mathcal{U}$.

Definition 4.1 (Pareto Optimality). The utility vector $\boldsymbol{R} \in \mathcal{U}$ is said to be *Pareto optimal* if and only if there is no other utility vector $\boldsymbol{R}' \in \mathcal{U}$ ($\boldsymbol{R}' \neq \boldsymbol{R}$) such that $R'_i \leq R_i, \forall i \in \mathcal{N}$.

That is to say, there is no other utility vector that leads to superior performances for some players without causing inferior performances for other players.

In our work, we assume that the utility vectors which satisfy the performance requirements of all cloud users exist, i.e., the set

$$\mathcal{A} = \{\boldsymbol{R} \,|\boldsymbol{R} \in \mathcal{U} \text{ and } R_i \leq R_{i,\max}, \forall i \in \mathcal{N}\} \tag{11}$$

is nonempty. Let $\mathcal{P} = \{(\mathcal{U}, \boldsymbol{R}_{\max}) \,|\mathcal{U} \subset \Re^n\}$ be the pair set of achievable performance with respect to the initial performance requirements and S be a mapping from $\mathcal{P}$ to $\Re^n$, i.e., $S : \mathcal{P} \to \Re^n$. Then, with reference to Penmatsa and Chronopoulos (2011) and Yaïche et al. (2000), we get the definition of Nash bargaining solution as follows.

Definition 4.2 (Nash Bargaining Solution). $S(\mathcal{U}, \boldsymbol{R}_{\max})$ ($(\mathcal{U}, \boldsymbol{R}_{\max}) \in \mathcal{P}$) is said to be a *Nash bargaining solution* if $S(\mathcal{U}, \boldsymbol{R}_{\max}) \in \mathcal{A}$, $S(\mathcal{U}, \boldsymbol{R}_{\max})$ is Pareto optimal and satisfies three axioms, i.e., linear axiom, irrelevant alternatives axiom, and symmetry axiom.

4.2 NBS Existence Analysis

In this subsection, we analyze the existence of NBS for the formulated game $\mathcal{G} = \langle \mathcal{Q}, \boldsymbol{R} \rangle$. Before addressing the bargaining solution existence analysis, we first show a property presented in Theorem 3.1, which is helpful to prove the existence of NBS for the game $\mathcal{G}$.

Theorem 4.3. *Given a fixed $\boldsymbol{\lambda}_{-i}$ ($i \in \mathcal{N}$), which is the request allocation strategies of all users except that of user i, i.e., $\boldsymbol{\lambda}_{-i} = (\boldsymbol{\lambda}_k)_{k=1,k\neq i}^{n}$, then the utility function $R_i(\boldsymbol{\lambda})$ is convex in $\boldsymbol{\lambda}_i$, where $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_{-i}) \in \mathcal{Q}$ with $\mathcal{Q}$ defined by the constraints in Equation (3).*

Proof. According to the results in Liu et al. (2014) and Scutari et al. (2012), we know that the proof of the above theorem follows if we can show that the Hessian matrix of the utility function

$R_i(\boldsymbol{\lambda})$ is positive semidefinite. Since

$$R_i(\boldsymbol{\lambda}) = \frac{1}{\Lambda_i}\sum_{j=1}^{m}\left(\frac{\lambda_{ij}}{\mu_j} + \frac{\lambda_{ij}}{c_j\mu_j - \chi_j}\right),$$

with χ_j denoting the aggregated requests on server j ($j \in \mathcal{M}$), i.e., $\chi_j = \sum_{k=1}^{n}\lambda_{kj}$, we have

$$\begin{aligned}\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}} &= \frac{\partial}{\partial \lambda_{ij}}\left(\lambda_{ij}T_j(\boldsymbol{\lambda})\right)\\ &= \frac{1}{\Lambda_i}\left(\frac{1}{\mu_j} + \frac{c_j\mu_j - \chi_j^{-i}}{\left(c_j\mu_j - \chi_j\right)^2}\right),\end{aligned}$$

where χ_j^{-i} denotes the aggregated requests on server j except that of user i, i.e., $\chi_j^{-i} = \sum_{k=1,k\neq i}^{n}\lambda_{kj}$.

We can further obtain

$$\frac{\partial^2 R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}^2} = \frac{2\left(c_j\mu_j - \chi_j^{-i}\right)}{\Lambda_i\left(c_j\mu_j - \chi_j\right)^3},$$

and

$$\frac{\partial^2 R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}\partial \lambda_{ik}} = 0,$$

where $j, k \in \mathcal{M}$ and $j \neq k$. Then we have

$$\nabla_{\boldsymbol{\lambda}_i} R_i(\boldsymbol{\lambda}) = \left[\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}}\right]_{j=1}^{m} = \left(\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{i1}}, \ldots, \frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{im}}\right),$$

and the Hessian matrix is expressed as

$$\begin{aligned}\nabla^2_{\boldsymbol{\lambda}_i} R_i(\boldsymbol{\lambda}) &= \operatorname{diag}\left\{\left[\frac{\partial^2 R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}^2}\right]_{j=1}^{m}\right\}\\ &= \operatorname{diag}\left\{\left[\frac{2\left(c_j\mu_j - \chi_j^{-i}\right)}{\Lambda_i\left(c_j\mu_j - \chi_j\right)^3}\right]_{j=1}^{m}\right\}. \qquad (12)\end{aligned}$$

Obviously, the diagonal matrix in Equation (12) has all diagonal elements being positive. Thus, the Hessian matrix of $R_i(\boldsymbol{\lambda})$ is positive semidefinite. This completes the proof and the result follows.

Theorem 4.4. *There exists a NBS for the formulated cooperative game* $\mathcal{G} = \langle \boldsymbol{Q}, \boldsymbol{R}\rangle$.

Proof. According to the Theorem 2.1 in Yaïche et al. (2000), we know that the above claim follows if two conditions are satisfied. First, each utility function $R_i(\boldsymbol{\lambda})$ ($i \in \mathcal{N}$) is convex on $\boldsymbol{Q}$. Second, the request allocation strategy set $\boldsymbol{Q}$ is convex and compact. Obviously, the strategy set $\boldsymbol{Q}$ is compact. By Theorem 4.3, we also know that the first condition is satisfied. Therefore, we only need to show the convexity of the strategy set $\boldsymbol{Q}$.

To prove the convexity of the strategy set $\boldsymbol{Q}$, it suffices to show that $\forall \boldsymbol{\lambda}, \boldsymbol{s} \in \boldsymbol{Q}$, the vector $\boldsymbol{t} = \theta\boldsymbol{\lambda} + (1-\theta)\boldsymbol{s} \in \boldsymbol{Q}$, where $\theta \in [0,1]$. Since $\boldsymbol{\lambda}, \boldsymbol{s} \in \boldsymbol{Q}$, we have

$$\begin{cases}\sum_{j=1}^{m}\lambda_{ij} = \Lambda_i, \forall i \in \mathcal{N},\\ \sum_{i=1}^{n}\lambda_{ij} \leq c_j\hat{\mu}_j, \forall j \in \mathcal{M},\\ \lambda_{ij} \geq 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M},\end{cases}$$

and

$$\begin{cases} \sum_{j=1}^{m} s_{ij} = \Lambda_i, \forall i \in \mathcal{N}, \\ \sum_{i=1}^{n} s_{ij} \leq c_j \hat{\mu}_j, \forall j \in \mathcal{M}, \\ s_{ij} \geq 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \end{cases}$$

where $\hat{\mu}_j = \mu_j - \epsilon$ with ϵ denoting a very small positive constant. Hence, we obtain

$$\begin{aligned} \sum_{j=1}^{m} t_{ij} &= \sum_{j=1}^{m} \left(\theta \lambda_{ij} + (1-\theta)\, s_{ij}\right) \\ &= \theta \sum_{j=1}^{m} \lambda_{ij} + (1-\theta) \sum_{j=1}^{m} s_{ij} \\ &= \theta \Lambda_i + (1-\theta)\, \Lambda_i = \Lambda_i, \end{aligned}$$

for all $i \in \mathcal{N}$, and

$$\begin{aligned} \sum_{i=1}^{n} t_{ij} &= \sum_{i=1}^{n} (\theta \lambda_{ij} + (1-\theta)\, s_{ij}) \\ &= \theta \sum_{i=1}^{n} \lambda_{ij} + (1-\theta) \sum_{i=1}^{n} s_{ij} \\ &\leq \theta c_j \hat{\mu}_j + (1-\theta)\, c_j \hat{\mu}_j = c_j \hat{\mu}_j, \end{aligned}$$

for all $j \in \mathcal{M}$. In addition, $t_{ij} \geq 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}$. Therefore, $\boldsymbol{t} = \theta \boldsymbol{\lambda} + (1-\theta)\, \boldsymbol{s}$ is also in $\mathcal{Q}$. This completes the proof and the result follows. □

4.3 NBS Computation Analysis

Once we have established the existence of NBS for the formulated game $\mathcal{G} = \langle \mathcal{Q}, \boldsymbol{R} \rangle$, we are interested in obtaining a suitable algorithm to compute the NBS.

Motivated by Yaïche et al. (2000) and Guo et al. (2013), we know that we can compute the NBS of game $\mathcal{G}$ by solving the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & G_{\Pi}(\boldsymbol{\lambda}) = \prod_{i=1}^{n} (R_{i,\max} - R_i(\boldsymbol{\lambda})), \\ \text{s.t.} \quad & \boldsymbol{\lambda} \in \mathcal{Q}. \end{aligned} \tag{13}$$

In the bargaining game, multiple players enter the game with initial performance requirements as well as a utility function. They cooperate to achieve a win-win situation, in which the social gains (represented by the product in Equation (13)) are maximized. This corresponds to the scheduling strategy in cloud computing, with respect to guaranteeing the average response time demands for all players. By taking the logarithm of the objective, we can derive an equivalent optimization problem (P_r):

$$\begin{aligned} \text{maximize} \quad & G_{\Sigma}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \ln (R_{i,\max} - R_i(\boldsymbol{\lambda})), \\ \text{s.t.} \quad & \boldsymbol{\lambda} \in \mathcal{Q}. \end{aligned} \tag{14}$$

Namely,

$$\begin{aligned} \text{maximize} \quad & G_{\Sigma}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \ln\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right), \\ \text{s.t.} \quad & \sum_{j=1}^{m} \lambda_{ij} = \Lambda_i, \forall i \in \mathcal{N}, \\ & \sum_{i=1}^{n} \lambda_{ij} \le c_j \hat{\mu}_j, \forall j \in \mathcal{M}, \\ & \lambda_{ij} \ge 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \end{aligned} \tag{15}$$

where $\hat{\mu}_j = \mu_j - \epsilon$, with ϵ denoting a very small positive constant.

Theorem 4.5. *There exist $\alpha_i > 0$ and $\gamma_{ij} \ge 0$ ($i \in \mathcal{N}, j \in \mathcal{M}$) such that*

$$\frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})} \left(-\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_i^j}\right) = -\alpha_i + \gamma_{ij}, \tag{16}$$

for all $i \in \mathcal{N}, j \in \mathcal{M}$, and

$$\begin{cases} \sum_{j=1}^{m} \lambda_{ij} - \Lambda_i = 0, \forall i \in \mathcal{N}, \\ \gamma_i^j \lambda_i^j = 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \end{cases} \tag{17}$$

where $\boldsymbol{\lambda}$ is a Nash bargaining solution for the optimization problem (P_r).

Proof. We can maximize $G_{\Sigma}(\boldsymbol{\lambda})$ in Equation (15) by using the method of Lagrange multiplier. Let α_i ($\forall i \in \mathcal{N}$), β_j ($\forall j \in \mathcal{M}$), and γ_{ij} ($\forall i \in \mathcal{N}, \forall j \in \mathcal{M}$) be the Lagrange multipliers for the individual request constraint, server processing capacity constraint, and lower bound allocation constraint in Equation (15), respectively. Then, the Lagrangian of the optimization problem in Equation (15) is

$$\begin{aligned} L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = & \sum_{i=1}^{n} \ln\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right) \\ & + \sum_{i=1}^{n} \alpha_i \left(\sum_{j=1}^{m} \lambda_{ij} - \Lambda_i\right) + \sum_{j=1}^{m} \beta_j \left(\sum_{i=1}^{n} \lambda_{ij} - c_j \hat{\mu}_j\right) - \sum_{i=1}^{n} \sum_{j=1}^{m} \gamma_{ij} \lambda_{ij}, \end{aligned}$$

where $\boldsymbol{\alpha} = (\alpha_i)_{i \in \mathcal{N}}$, $\boldsymbol{\beta} = (\beta_j)_{j \in \mathcal{M}}$, and $\boldsymbol{\gamma} = (\gamma_{ij})_{i \in \mathcal{N}, j \in \mathcal{M}}$. The optimal solution satisfies the following Kuhn-Tucker conditions:

$$\frac{\partial L}{\partial \lambda_{ij}} = \frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})} \left(-\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}}\right) + \alpha_i + \beta_j - \gamma_{ij} = 0,$$

for all $i \in \mathcal{N}, j \in \mathcal{M}$, and

$$\begin{cases} \sum_{j=1}^{m} \lambda_{ij} - \Lambda_i = 0, \forall i \in \mathcal{N}, \\ \beta_j \left(\sum_{i=1}^{n} \lambda_{ij} - c_j \hat{\mu}_j\right) = 0, \forall j \in \mathcal{M}, \\ \gamma_{ij} \lambda_{ij} = 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \end{cases}$$

where $\hat{\mu}_j = \mu_j - \epsilon$, with ϵ denoting a very small positive constant.

Notice that, if the aggregated requests on server j ($j \in \mathcal{M}$) reach its upper bound constraint, i.e., $\sum_{i=1}^{n} \lambda_{ij} = c_j \hat{\mu}_j$, then the average response time of server j is very large. This can cause two results.

First, the average response times of some players exceed their performance requirements. Second, the average response times of some players are very close to their performance requirements, which obviously cannot maximize the value of $G_\Sigma(\boldsymbol{\lambda})$. Therefore, at a NBS, the aggregated requests on each machine are strictly less than the processing capacity of the server, i.e., $\sum_{i=1}^{n}\lambda_{ij} < c_j\hat{\mu}_j$ $(\forall j \in \mathcal{M})$. This implies that $\beta_j = 0$ $(\forall j \in \mathcal{M})$, and the above Kuhn-Tucker conditions are equivalent to the following constraints:

$$\frac{\partial L}{\partial \lambda_i^j} = \frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})}\left(-\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}}\right) = -\alpha_i + \gamma_{ij},$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, and

$$\begin{cases} \sum_{j=1}^{m}\lambda_{ij} - \Lambda_i = 0, \forall i \in \mathcal{N}, \\ \gamma_{ij}\lambda_{ij} = 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}. \end{cases}$$

This complements the proof and the result follows. □

So far, to obtain the optimal solution, i.e., obtain the NBS, we have shown how to achieve our scheduling goal. We have to solve the equations presented by Equations (16) and (17) with $(2nm + n)$ variables, whose computation is very complex. In Section 4.4, we consider a special situation, in which all allocating substreams, i.e., λ_{ij} $(\forall i \in \mathcal{N}, \forall j \in \mathcal{M})$, are strictly positive. For this situation, we propose a $C\mathcal{A}$, which can find the NBS very efficiently. In Section 4.5, we consider the more general case, and propose an $\mathcal{IA}$, which is based on duality theory.

4.4 A Special Situation

In this section, we focus on the calculation for the NBS when all allocating substreams are strictly positive, i.e., $\lambda_{ij} > 0$ $(\forall i \in \mathcal{N}, \forall j \in \mathcal{M})$. By Equations (16) and (17) in Theorem 4.5, we know that if $\lambda_{ij} > 0$ for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, then we have $\gamma_{ij} = 0$ $(\forall i \in \mathcal{N}, \forall j \in \mathcal{M})$, and we can obtain

$$\frac{\partial L}{\partial \lambda_{ij}} = \frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})}\left(-\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}}\right) = -\alpha_i, \tag{18}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$. That is,

$$\begin{aligned} \frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}} &= \frac{1}{\Lambda_i}\left(\frac{1}{\mu_j} + \frac{c_j\mu_j - \chi_j^{-i}}{(c_j\mu_j - \chi_j)^2}\right) \\ &= \alpha_i\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right), \end{aligned} \tag{19}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, where χ_j denotes the aggregated requests on server j, i.e., $\chi_j = \sum_{k=1}^{n}\lambda_{kj}$, and χ_j^{-i} denotes the aggregated requests on server j except that of user i, i.e., $\chi_j^{-i} = \sum_{k=1,k\neq i}^{n}\lambda_{ij}$. We can further obtain

$$\frac{1}{\mu_j} + \frac{c_j\mu_j - \chi_j^{-i}}{(c_j\mu_j - \chi_j)^2} = \phi_i, \tag{20}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, with $\phi_i = \alpha_i\Lambda_i\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right)$.

Add up the left hand and right hand of Equation (20) of all cloud users, respectively. We get

$$\frac{n}{\mu_j} + \frac{nc_j\mu_j - \sum_{i=1}^{n}\chi_j^{-i}}{(c_j\mu_j - \chi_j)^2} = \phi_\Sigma, \tag{21}$$

for all $j \in \mathcal{M}$, where $\phi_\Sigma = \sum_{i=1}^{n} \phi_i$, $\chi_j = \sum_{i=1}^{n} \lambda_{ij}$, and $\chi_j^{-i} = \chi_j - \lambda_{ij}$. Namely,

$$\frac{n}{\mu_j} + \frac{nc_j\mu_j - (n-1)\,\chi_j}{(c_j\mu_j - \chi_j)^2} = \phi_\Sigma, \tag{22}$$

for all $j \in \mathcal{M}$. We have

$$\chi_j = c_j\mu_j - \frac{(n-1) + \sqrt{(n-1)^2 + 4\left(\phi_\Sigma - \frac{n}{\mu_j}\right)c_j\mu_j}}{2\left(\phi_\Sigma - \frac{n}{\mu_j}\right)}, \tag{23}$$

for all $j \in \mathcal{M}$. Since the summation of the aggregated requests on all servers is equal to the total arrival requests of all cloud users, i.e., $\sum_{j=1}^{m} \chi_j = \sum_{i=1}^{n} \Lambda_i$, we obtain

$$\sum_{j=1}^{m}\left(\frac{(n-1) + \sqrt{(n-1)^2 + 4\left(\phi_\Sigma - \frac{n}{\mu_j}\right)c_j\mu_j}}{2\left(\phi_\Sigma - \frac{n}{\mu_j}\right)}\right) = P_{\mathcal{M}}, \tag{24}$$

with

$$P_{\mathcal{M}} = \sum_{j=1}^{m} c_j\mu_j - \sum_{i=1}^{n} \Lambda_i. \tag{25}$$

Obviously, it is hard to directly derive the value of ϕ_Σ according to Equation (24).

However, in view of Equation (24), we can rewrite the equation as the following:

$$\sum_{j=1}^{m}\left(\frac{(n-1)}{2\left(\phi_\Sigma - \frac{n}{\mu_j}\right)} + \sqrt{\frac{(n-1)^2}{4\left(\phi_\Sigma - \frac{n}{\mu_j}\right)^2} + \frac{c_j\mu_j}{\left(\phi_\Sigma - \frac{n}{\mu_j}\right)}}\right) = P_{\mathcal{M}}. \tag{26}$$

It is easy to observe that the left-hand side of Equation (26) decreases with the increase of ϕ_Σ. Let

$$f(\phi_\Sigma) = \sum_{j=1}^{m}\left(\frac{(n-1) + \sqrt{(n-1)^2 + 4\left(\phi_\Sigma - \frac{n}{\mu_j}\right)c_j\mu_j}}{2\left(\phi_\Sigma - \frac{n}{\mu_j}\right)}\right). \tag{27}$$

We can conclude that $f(\phi_\Sigma)$ decreases with the increase of ϕ_Σ. Hence, we can use a binary search method to calculate the value of ϕ_Σ, which satisfies Equation (24).

After ϕ_Σ is calculated, we can calculate the aggregated requests on each server j as Equation (23), i.e., χ_j ($\forall j \in \mathcal{M}$). Then, we can calculate the request allocation strategy of each user i ($i \in \mathcal{N}$) according to Equation (20). Since $\chi_j^{-i} = \chi_j - \lambda_{ij}$, we have

$$\frac{1}{\mu_j} + \frac{c_j\mu_j - \chi_j + \lambda_{ij}}{(c_j\mu_j - \chi_j)^2} = \phi_i, \tag{28}$$

and

$$\lambda_{ij} = \left(\phi_i - \frac{1}{\mu_j}\right)(c_j\mu_j - \chi_j)^2 - (c_j\mu_j - \chi_j), \tag{29}$$

for all $j \in \mathcal{M}$. Since $\sum_{j=1}^{m} \lambda_{ij} = \Lambda_i$, we obtain

$$\phi_i = \frac{\Lambda_i + \sum_{j=1}^{m}\left((c_j\mu_j - \chi_j)\left(\frac{c_j\mu_j - \chi_j}{\mu_j} + 1\right)\right)}{\sum_{j=1}^{m}(c_j\mu_j - \chi_j)^2}. \tag{30}$$

ALGORITHM 1: *C*omputational *A*lgorithm ($\mathcal{CA}$)

Require: $\boldsymbol{\Lambda}$, $\boldsymbol{R}_{\max}$, ϵ.
Ensure: $\boldsymbol{\lambda}$.
1: //*Find a proper upper bound for* ϕ_Σ
2: Set $lb \leftarrow \max_{j\in\mathcal{M}}(n/\mu_j)$, and $\phi_\Sigma \leftarrow lb + inc$.
3: **while** $(f(\phi_\Sigma) > P_\mathcal{M})$ **do**
4: Set $inc \leftarrow 2 \times inc$, $lb \leftarrow \phi_\Sigma$, and $\phi_\Sigma \leftarrow \max_{j\in\mathcal{M}}(n/\mu_j) + inc$.
5: **end while**
6: Set $ub \leftarrow \max_{j\in\mathcal{M}}(n/\mu_j) + inc$.
7: //*Using binary search method search* ϕ_Σ
8: **while** $(ub - lb > \epsilon)$ **do**
9: Set $mid \leftarrow (ub + lb)/2$, and $\phi_\Sigma \leftarrow mid$.
10: **if** $(f(\phi_\Sigma) > P_\mathcal{M})$ **then**
11: Set $lb \leftarrow mid$.
12: **else**
13: Set $ub \leftarrow mid$.
14: **end if**
15: **end while**
16: Set $\phi_\Sigma \leftarrow (ub + lb)/2$.
17: //*Calculate the allocation strategy* $\boldsymbol{\lambda}$
18: **for** $(j \leftarrow 1$ to $m)$ **do**
19: Calculate χ_j as Equation (23).
20: **end for**
21: **for** $(i \leftarrow 1$ to $n)$ **do**
22: Calculate ϕ_i as Equation (30).
23: **end for**
24: **for** $(i \leftarrow 1$ to $n)$ **do**
25: **for** $(j \leftarrow 1$ to $m)$ **do**
26: Calculate λ_{ij} as Equation (29).
27: **end for**
28: **end for**
29: **return** $\boldsymbol{\lambda}$.

Substituting the above equation to Equation (29), we get the request allocation strategy for cloud user i on server j, i.e., λ_{ij}. Then, we can obtain the request allocation strategies $\boldsymbol{\lambda}$ of all cloud users, where $\boldsymbol{\lambda} = (\boldsymbol{\lambda}_i)_{i=1}^n$ with $\boldsymbol{\lambda}_i = (\lambda_{ij})_{j=1}^m$. The idea is formalized in Algorithm 1.

Given $\boldsymbol{\Lambda}$, $\boldsymbol{R}_{\max}$, and ϵ, where $\boldsymbol{\Lambda}$ is the request arrival rate vector, i.e., $\boldsymbol{\Lambda} = (\Lambda_i)_{i=1}^n$, $\boldsymbol{R}_{\max}$ is the average response time requirement vector of all cloud users, i.e., $\boldsymbol{R}_{\max} = (R_{i,\max})_{i=1}^n$, and ϵ is a very small constant. The $\mathcal{CA}$ finds the NBS, which can simultaneously satisfy the average response time demands of all users. The value of ϕ_Σ can be found by using the binary search method (Steps 8–16). The search interval $[lb, ub]$ for ϕ_Σ is determined as follows. For ub, we notice that the left-hand side of Equation (24) is an increasing function of ϕ_Σ (see Equation (27)). Then, we set an increment variable inc, which is initialized as a relative small positive constant and repeatedly doubled (Step 4). The value of inc is added to $\max_{j\in\mathcal{M}}(n/\mu_j)$ until the function value $f(\phi_\Sigma)$ is at least $P_\mathcal{M}$ (Steps 3–5). As for lb, notice that it must be greater than $\max_{j\in\mathcal{M}}(n/\mu_j)$. Therefore, we set lb as $(\max_{j\in\mathcal{M}}(n/\mu_j) + inc)$ (Step 2). Once $[lb, ub]$ is decided, ϕ_Σ can be searched based on the

fact that $f(\phi_\Sigma)$ is an increasing function of ϕ_Σ. After ϕ_Σ is determined, the aggregated requests on each server j, i.e., χ_j ($j \in \mathcal{M}$), can be computed (Steps 18–20). Then, the value ϕ_i ($i \in \mathcal{N}$) for each cloud user i can be calculated (Steps 21–23). Finally, the scheduling strategy of all cloud users can be computed based on previous calculations (Steps 24–28).

By Algorithm 1 ($C\mathcal{A}$ algorithm), we note that the most time-consuming process is the two while loops (Steps 3–5 and Steps 8–15). The fist while loop (Steps 3–5) requires time complexity $\Theta(\log \frac{ub-lb}{inc})$, where inc is the presented relative small constant, $lb = \max_{j\in\mathcal{M}}(n/\mu_j)$, and ub is the obtained value which satisfies $f(ub) < P_{\mathcal{M}}$. The second while loop (Steps 8–15) requires time complexity $\Theta(\log \frac{ub-lb}{\epsilon})$, where ϵ is the tolerated error. Since ϵ is usually smaller than inc, the time complexity of the $C\mathcal{A}$ algorithm is $\Theta(\log \frac{ub-lb}{\epsilon})$, and if we can evaluate the value of ub, we obtain the time complexity. From Equation (22), we have

$$\phi_\Sigma = \frac{n}{\mu_j} + \frac{nc_j\mu_j - (n-1)\,\chi_j}{(c_j\mu_j - \chi_j)^2} < \frac{n}{\mu_j} + \frac{nc_j\mu_j}{(c_j\mu_j - \chi_j)^2} < \frac{n}{\mu_j} + \frac{nc_j\mu_j}{\epsilon^2}. \tag{31}$$

Let $c_{\max} = \max_{j\in\mathcal{M}}(c_j)$, $\mu_{\min} = \min_{j\in\mathcal{M}}(\mu_j)$, and $\mu_{\max} = \max_{j\in\mathcal{M}}(\mu_j)$. Then, $lb = n/\mu_{\min}$, and ub is bounded by $(\frac{n}{\mu_{\min}} + \frac{nc_{\max}\mu_{\max}}{\epsilon^2})$. Therefore, the time complexity of the $C\mathcal{A}$ algorithm is $\Theta(\log \frac{nc_{\max}\mu_{\max}}{\epsilon^3})$.

4.5 An Iterative Algorithm

In this section, we consider the NBS computation for the general case, i.e., we consider boundaries. We propose an $\mathcal{IA}$, which is based on duality theory. The convergency of our proposed algorithm is also analyzed.

4.5.1 Dual-Based Decomposition. We first define a primal problem that has the same optimal solution as Equation (15) and then obtain the dual problem corresponding to the primal problem with no duality gap. In view of Equation (15), we can rewrite it as follows:

$$\begin{aligned} \text{minimize} \quad & G_\Sigma^-(\boldsymbol{\lambda}) = -\sum_{i=1}^{n} \ln\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right), \\ \text{s.t.} \quad & \sum_{j=1}^{m} \lambda_{ij} = \Lambda_i, \forall i \in \mathcal{N}, \\ & \sum_{i=1}^{n} \lambda_{ij} \le c_j\hat{\mu}_j, \forall j \in \mathcal{M}, \\ & \lambda_{ij} \ge 0, \forall i \in \mathcal{N}, \forall j \in \mathcal{M}. \end{aligned} \tag{32}$$

The Lagrangian function associated with the primal problem (32) is defined as

$$\begin{aligned} L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\gamma}) = & -\sum_{i=1}^{n} \ln\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right) \\ & - \sum_{i=1}^{n} \alpha_i \left(\sum_{j=1}^{m} \lambda_{ij} - \Lambda_i\right) + \sum_{i=1}^{n}\sum_{j=1}^{m} \gamma_{ij}\lambda_{ij}, \end{aligned} \tag{33}$$

where $\boldsymbol{\alpha} = (\alpha_i)_{i\in\mathcal{N}}$, and $\boldsymbol{\gamma} = (\gamma_{ij})_{i\in\mathcal{N}, j\in\mathcal{M}}$ are the Lagrange multipliers.

Notice that $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are also the dual variables associated with the primal problem. The dual function $d : \Re^n \times \Re^{n\times m} \to \Re$ corresponding to $L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\gamma})$ is expressed as

$$d(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \inf_{\boldsymbol{\lambda}\in\Re^{n\times m}} L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\gamma}). \tag{34}$$

Since the primal problem has a unique optimal solution, the dual function yields lower bound on the optimal $\boldsymbol{\lambda}^*$ which solves Equation (34). For any $\boldsymbol{\alpha} \in \Re^n$, $\boldsymbol{\gamma} \in \Re^{n\times m}$, we have $d(\boldsymbol{\alpha}, \boldsymbol{\gamma}) \leq L(\boldsymbol{\lambda}^*, \boldsymbol{\alpha}, \boldsymbol{\gamma})$. Because Q is convex and G_{Σ}^{-} is convex over Q, the Slater's condition holds, which is a sufficient condition for strong duality (Boyd and Vandenberghe 2004). Hence, there is no duality gap, and there exist $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ satisfying $d(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = L(\boldsymbol{\lambda}^*, \boldsymbol{\alpha}, \boldsymbol{\gamma})$.

To conclude, we obtain the dual problem corresponding to the primal problem with no duality gap. The dual problem (P_d) is described as follows:

$$\max_{\boldsymbol{\alpha}\in\Re^n, \boldsymbol{\gamma}\in\Re^{n\times m}} d(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = L(\boldsymbol{\lambda}^*, \boldsymbol{\alpha}, \boldsymbol{\gamma}), \tag{35}$$

where $d(\boldsymbol{\alpha}, \boldsymbol{\gamma})$ is the dual function and $L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\gamma})$ is the Lagrangian of the primal problem.

4.5.2 Gradient Method for the Dual Problem. To solve the primal problem, we first obtain the solution to the dual problem. By using a suitable step size, we design an iterative algorithm and update the variables $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ by applying the subgradient method (Boyd and Vandenberghe 2004). We define the following recursion:

$$\alpha_i^{(k+1)} = \alpha_i^{(k)} + \xi_\alpha \frac{\partial d}{\partial \alpha_i}, \tag{36}$$

for all $i \in \mathcal{N}$, and

$$\gamma_{ij}^{(k+1)} = \max\left(0, \gamma_{ij}^{(k)} + \xi_\gamma \frac{\partial d}{\partial \gamma_{ij}}\right), \tag{37}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, where ξ_α and ξ_γ are the step sizes. Obviously,

$$\frac{\partial d}{\partial \alpha_i} = \sum_{j=1}^{m} \bar{\lambda}_{ij} - \Lambda_i, \tag{38}$$

and

$$\frac{\partial d}{\partial \gamma_{ij}} = -\bar{\lambda}_{ij}, \tag{39}$$

where $\bar{\boldsymbol{\lambda}}$ denotes the optimal scheduling strategy which minimizes the Lagrangian $L(\bar{\boldsymbol{\lambda}}, \boldsymbol{\alpha}, \boldsymbol{\gamma})$.

Motivated by Boyd and Vandenberghe (2004), we choose step sizes according to the *diminishing step size rules*, i.e.,

$$\sum_{k=1}^{\infty} \xi_l^{(k)} = \infty, \lim_{k\to\infty} \xi_l^{(k)} = 0, \forall l \in \{\alpha, \gamma\}. \tag{40}$$

At each iteration k, if the variables $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ are determined, then we can find the scheduling strategy $\boldsymbol{\lambda}$ corresponding to the dual function:

$$d(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = \inf_{\boldsymbol{\lambda}\in\Re^{n\times m}} L(\boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\gamma}). \tag{41}$$

Namely, find a strategy

$$\boldsymbol{\lambda} \in \arg\min_{\boldsymbol{\lambda}'\in\Re^{n\times m}} L(\boldsymbol{\lambda}', \boldsymbol{\alpha}, \boldsymbol{\gamma}). \tag{42}$$

Then, we have

$$\frac{\partial L}{\partial \lambda_{ij}} = -\frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})} \cdot \left(-\frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}}\right) - \alpha_i + \gamma_{ij} = 0, \tag{43}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$. We obtain

$$\frac{1}{R_{i,\max} - R_i(\boldsymbol{\lambda})} \cdot \frac{\partial R_i(\boldsymbol{\lambda})}{\partial \lambda_{ij}} = \phi_{ij}, \tag{44}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$, where $\phi_{ij} = \alpha_i - \gamma_{ij}$. That is,

$$\frac{1}{\Lambda_i}\left(\frac{1}{\mu_j} + \frac{c_j\mu_j - \chi_j + \lambda_{ij}}{(c_j\mu_j - \chi_j)^2}\right) = \phi_{ij}\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right)$$
$$= \phi_{ij}\left(R_{i,\max} - \frac{1}{\Lambda_i}\sum_{l=1}^{m}\left(\frac{\lambda_{il}}{\mu_l} + \frac{\lambda_{il}}{c_l\mu_l - \chi_l}\right)\right), \tag{45}$$

for all $i \in \mathcal{N}$, $j \in \mathcal{M}$.

Notice that the nonlinear equation set (45) has nm variables, which is very hard to solve when the size of nm is somewhat large. To solve the equation set, we first find an equivalent form, which has only m variables, and then try to solve it. Specifically, denote r_j as the remaining processing capacity of server j, i.e., $r_j = c_j\mu_j - \chi_j$ ($j \in \mathcal{M}$), then we find an equivalent form of Equation (45), which has only the m variables r_j ($j \in \mathcal{M}$).

THEOREM 4.6. *The solution to equation set (45) is equivalent to that of the equation set* $\mathbf{F}(\boldsymbol{r}) = \mathbf{0}$, *where*

$$\mathbf{F}(\boldsymbol{r}) = (F_j(r_j, \boldsymbol{r}_{-j}))_{j=1}^{m}, \tag{46}$$

with

$$F_j(r_j, \boldsymbol{r}_{-j}) = c_j\mu_j - \sum_{i=1}^{n}\frac{f_{ij}(r_j, \boldsymbol{r}_{-j})}{h_{ij}(r_j, \boldsymbol{r}_{-j})} - r_j, \tag{47}$$

and

$$f_{ij}(r_j, \boldsymbol{r}_{-j}) = \phi_{ij}\Lambda_i R_{i,\max} - \left(\frac{1}{\mu_j} + \frac{1}{r_j}\right)$$
$$- \phi_{ij}\sum_{l=1, l\neq j}^{m}\left[r_l^2\left(\frac{\phi_{il}}{\phi_{ij}\mu_j} + \frac{\phi_{il}}{\phi_{ij}r_j} - \frac{1}{\mu_l} - \frac{1}{r_l}\right)\left(\frac{1}{\mu_l} + \frac{1}{r_l}\right)\right], \tag{48}$$

and

$$h_{ij}(r_j, \boldsymbol{r}_{-j}) = \frac{1}{r_j^2} + \frac{\phi_{ij}}{r_j} + \frac{\phi_{ij}}{\mu_j} + \phi_{ij}\sum_{l=1, l\neq j}^{m}\frac{\phi_{il}r_l^2}{\phi_{ij}r_j^2}\left(\frac{1}{\mu_l} + \frac{1}{r_l}\right), \tag{49}$$

where $\boldsymbol{r}_{-j}$ *denotes the remaining processing capacities of all servers except that of server* j, *i.e.,* $\boldsymbol{r}_{-j} = (r_l)_{l=1, l\neq j}^{m}$.

PROOF. Since r_j is the remaining processing capacity of server j, i.e., $r_j = c_j\mu_j - \chi_j$ ($\forall j \in \mathcal{M}$), then, from Equation (45), we obtain

$$\frac{1}{\mu_j} + \frac{1}{r_j} + \frac{\lambda_{ij}}{r_j^2} = \phi_{ij}\Lambda_i\left(R_{i,\max} - R_i(\boldsymbol{\lambda})\right).$$

Since $\forall l, j \in \mathcal{M}$, the values of $(R_{i,\max} - R_i(\boldsymbol{\lambda}))$ are the same for a specific user i ($i \in \mathcal{N}$), we have

$$\frac{1}{\mu_l} + \frac{1}{r_l} + \frac{\lambda_{il}}{r_l^2} = \frac{\phi_{il}}{\phi_{ij}}\left(\frac{1}{\mu_j} + \frac{1}{r_j} + \frac{\lambda_{ij}}{r_j^2}\right),$$

and

$$\lambda_{il} = \frac{\phi_{il}r_l^2}{\phi_{ij}r_j^2}\lambda_{ij} + r_l^2\left(\frac{\phi_{il}}{\phi_{ij}\mu_j} + \frac{\phi_{il}}{\phi_{ij}r_j} - \frac{1}{\mu_l} - \frac{1}{r_l}\right).$$

Substituting the above equation into Equation (45), we get

$$\left(\frac{1}{r_j^2}+\frac{\phi_{ij}}{r_j}+\frac{\phi_{ij}}{\mu_j}+\phi_{ij}\sum_{l=1,l\neq j}^{m}\frac{\phi_{il}r_l^2}{\phi_{ij}r_j^2}\left(\frac{1}{\mu_l}+\frac{1}{r_l}\right)\right)\lambda_{ij}$$
$$+\phi_{ij}\sum_{l=1,l\neq j}^{m}\left[r_l^2\left(\frac{\phi_{il}}{\phi_{ij}\mu_j}+\frac{\phi_{il}}{\phi_{ij}r_j}-\frac{1}{\mu_l}-\frac{1}{r_l}\right)\left(\frac{1}{\mu_l}+\frac{1}{r_l}\right)\right]$$
$$=\phi_{ij}\Lambda_i R_{i,\max}-\left(\frac{1}{\mu_j}+\frac{1}{r_j}\right).$$

We can further obtain

$$\lambda_{ij}=\frac{f_{ij}(r_j,\boldsymbol{r}_{-j})}{h_{ij}(r_j,\boldsymbol{r}_{-j})},$$

where

$$f_{ij}(r_j,\boldsymbol{r}_{-j})=\phi_{ij}\Lambda_i R_{i,\max}-\left(\frac{1}{\mu_j}+\frac{1}{r_j}\right)$$
$$-\phi_{ij}\sum_{l=1,l\neq j}^{m}\left[r_l^2\left(\frac{\phi_{il}}{\phi_{ij}\mu_j}+\frac{\phi_{il}}{\phi_{ij}r_j}-\frac{1}{\mu_l}-\frac{1}{r_l}\right)\left(\frac{1}{\mu_l}+\frac{1}{r_l}\right)\right],$$

and

$$h_{ij}(r_j,\boldsymbol{r}_{-j})=\frac{1}{r_j^2}+\frac{\phi_{ij}}{r_j}+\frac{\phi_{ij}}{\mu_j}+\phi_{ij}\sum_{l=1,l\neq j}^{m}\frac{\phi_{il}r_l^2}{\phi_{ij}r_j^2}\left(\frac{1}{\mu_l}+\frac{1}{r_l}\right).$$

On the other hand, since the aggregated request on each server is equivalent to its total processing capacity minus its remaining processing capacity, i.e., $\chi_j=c_j\mu_j-r_j=\sum_{i=1}^{n}\lambda_{ij}$ $(j\in\mathcal{M})$, we have

$$c_j\mu_j-r_j=\sum_{i=1}^{n}\frac{f_{ij}(r_j,\boldsymbol{r}_{-j})}{h_{ij}(r_j,\boldsymbol{r}_{-j})}.$$

Namely,

$$c_j\mu_j-\sum_{i=1}^{n}\frac{f_{ij}(r_j,\boldsymbol{r}_{-j})}{h_{ij}(r_j,\boldsymbol{r}_{-j})}-r_j=0,$$

for all $j\in\mathcal{M}$. This completes the proof and the result follows. □

If the remaining capacities of servers, i.e., $\boldsymbol{r}=(r_j)_{j\in\mathcal{M}}$, are calculated, we can calculate each allocating substream as $\lambda_{ij}=f_{ij}(\boldsymbol{r})/h_{ij}(\boldsymbol{r})$ $(\forall i\in\mathcal{N},\forall j\in\mathcal{M})$ according to the derivations in Theorem 4.6, where $f_{ij}(\boldsymbol{r})$ and $h_{ij}(\boldsymbol{r})$ are presented in Equations (48) and (49), respectively. The idea is formalized in Algorithm 2.

Given $\boldsymbol{\Lambda}$, $\boldsymbol{R}_{\max}$, and ϵ, where $\boldsymbol{\Lambda}$ is the request arrival rate vector, i.e., $\boldsymbol{\Lambda}=(\Lambda_i)_{i=1}^{n}$, $\boldsymbol{R}_{\max}$ is the average response time requirement vector of all cloud users, i.e., $\boldsymbol{R}_{\max}=(R_{i,\max})_{i=1}^{n}$, and ϵ is a very small constant. The $\mathcal{IA}$ tries to find a NBS. At the beginning of the iterations, the Lagrange multipliers are randomly chosen from the positive real set. We use a variable k to index each of the iterations, which is initialized as zero. At the beginning of the iteration k, the algorithm uses another subalgorithm *Calculate_r* described in Algorithm 3, which, given $\boldsymbol{\Lambda}$, $\boldsymbol{R}_{\max}$, $\boldsymbol{\phi}$, and ϵ, finds the remaining processing capacities of all servers, where $\boldsymbol{\phi}=(\phi_{ij})_{i\in\mathcal{N},j\in\mathcal{M}}$ with $\phi_{ij}=\alpha_i-\gamma_{ij}$.

Notice that the computation for the equation set in Theorem 4.6 is still complex even though the number of variables is reduced to m. The algorithm (Algorithm 3) is used to solve the equation set $\mathbf{F}(\boldsymbol{r})=\mathbf{0}$, which is presented in Theorem 4.6. We also use an iterative process to obtain results

in this algorithm. The key motivation is that when given the remaining processing capacities of all other servers $\boldsymbol{r}_{-j}$ ($j \in \mathcal{M}$), the function $F_j(r_j, \boldsymbol{r}_{-j})$ decreases with the increase of r_j. At the beginning of the iterations, the remaining processing capacities of all servers are initialized as small positive values. We use a variable k to index each of the iterations, which is initialized as zero. At the beginning of the iteration k, we update the remaining processing capacities for all servers sequentially (Steps 5–16). When given the remaining processing capacities of other servers $\boldsymbol{r}_{-j}$, we can find r_j by using the binary search method in certain interval $[lb, ub]$ (Steps 7–15 in Algorithm 3). We set lb as a relative small constant, and ub as $c_j\hat{\mu}_j$. Once $[lb, ub]$ is decided, r_j can be searched based on the fact that $F_j(r_j, \boldsymbol{r}_{-j})$ is a decreasing function of r_j (Steps 7–15). The algorithm terminates when the remaining processing capacities of all servers are kept unchanged, i.e., $\|\boldsymbol{r}^{(k)} - \boldsymbol{r}^{(k-1)}\| \leq \epsilon$.

At each iteration k of Algorithm 2, after the remaining processing capacities of all servers ($\boldsymbol{r}$) are obtained, we can compute each allocating substream (Step 7) and then update the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ by using a gradient method (Steps 10–15). The algorithm (Algorithm 2) terminates when the number of iterations reaches the pre-set maximum value.

ALGORITHM 2: $\mathcal{I}$terative $\mathcal{A}$lgorithm ($\mathcal{IA}$)

Require: Λ, $R_{\max}$, ϵ.
Ensure: $\boldsymbol{\lambda}$.
1: //*Initialization*
2: Choose $\boldsymbol{\alpha}^{(0)}, \boldsymbol{\gamma}^{(0)} \in \Re_+$, calculate $\boldsymbol{\phi}$, and set $k \leftarrow 0$.
3: **while** ($k \leq K_{\max}$) **do**
4: Set $\boldsymbol{r}^{(k)} \leftarrow Calculate\_\boldsymbol{r}(\Lambda, \boldsymbol{R}_{\max}, \boldsymbol{\phi}, \epsilon)$.
5: **for** (each user $i \in \mathcal{N}$) **do**
6: **for** (each server $j \in \mathcal{M}$) **do**
7: Calculate $f_{ij}(r_j^{(k)}, \boldsymbol{r}_{-j}^{(k)})$, $h_{ij}(r_j^{(k)}, \boldsymbol{r}_{-j}^{(k)})$, and set $\lambda_{ij}^{(k)} \leftarrow f_{ij}(r_j^{(k)}, \boldsymbol{r}_{-j}^{(k)}) \Big/ h_{ij}(r_j^{(k)}, \boldsymbol{r}_{-j}^{(k)})$.
8: **end for**
9: **end for**
10: **for** (each cloud user $i \in \mathcal{N}$) **do**
11: Set $\alpha_i^{(k+1)} \leftarrow \alpha_i^{(k)} + \xi_\alpha^{(k)}(\sum_{j=1}^{m} \lambda_{ij}^{(k)} - \Lambda_i)$.
12: **for** (each server $j \in \mathcal{M}$) **do**
13: Set $\gamma_{ij}^{(k+1)} \leftarrow \max\left(0, \gamma_{ij}^{(k)} - \xi_\gamma^{(k)}\lambda_{ij}^{(k)}\right)$.
14: **end for**
15: **end for**
16: Set $k \leftarrow k + 1$.
17: **end while**
18: **return** $\boldsymbol{\lambda}^{(k)}$.

$Calculate\_\boldsymbol{r}$ is an iterative algorithm, in which a for loop (Steps 5–16) is nested in an outer while loop (Steps 3–18). The for loop has m iterations and in each iteration, the main operation is the binary search process (Steps 7–15), which requires time complexity $\Theta(\log \frac{c_{\max}\mu_{\max} - inc}{\epsilon})$, where $c_{\max}$ is the maximal number of cores of a server in $\mathcal{M}$, i.e., $c_{\max} = \max_{j \in \mathcal{M}}(c_j)$, $\mu_{\max}$ is the maximal processing capacity of a core of a server in $\mathcal{M}$, i.e., $\mu_{\max} = \max_{j \in \mathcal{M}}(\mu_j)$, and inc is an initialized relative small constant. Hence, the for loop requires $\Theta(m \log \frac{c_{\max}\mu_{\max} - inc}{\epsilon})$. Since the number of iterations of the outer while loop (Steps 3–18) cannot be evaluated, we cannot determine the time complexity of $Calculate\_\boldsymbol{r}$. However, we have analyzed its convergence in Theorem 4.7.

ALGORITHM 3: *Calculate_r*$(\Lambda, R_{\max}, \boldsymbol{\phi}, \epsilon)$

Require: Λ, $R_{\max}$, $\boldsymbol{\phi}$, ϵ.
Ensure: $\boldsymbol{r}$.
1: //*Initialization*
2: Let *inc* be a relative small positive constant. Set $r_j^{(0)} \leftarrow inc$ $(\forall j \in \mathcal{M})$, $lb \leftarrow inc$, and $k \leftarrow 0$.
3: **while** $(\|\boldsymbol{r}^{(k)} - \boldsymbol{r}^{(k-1)}\| > \epsilon)$ **do**
4: Set $\boldsymbol{r}^{(c)} \leftarrow \boldsymbol{r}^{(k)}$.
5: **for** (each server $j \in \mathcal{M}$) **do**
6: Set $r_j^{(c)} \leftarrow lb + inc$, and $ub \leftarrow c_j\hat{\mu}_j$.
7: **while** $(ub - lb > \epsilon)$ **do**
8: Set $mid \leftarrow (ub + lb)/2$, and $r_j^{(c)} \leftarrow mid$.
9: **if** $(F_j(r_j^{(c)}, \boldsymbol{r}_{-j}^{(c)}) > 0)$ **then**
10: Set $lb \leftarrow mid$.
11: **else**
12: Set $ub \leftarrow mid$.
13: **end if**
14: Set $r_j^{(c)} \leftarrow (ub + lb)/2$.
15: **end while**
16: **end for**
17: Set $\boldsymbol{r}^{(k)} \leftarrow \boldsymbol{r}^{(c)}$, and $k \leftarrow k + 1$.
18: **end while**
19: **return** $\boldsymbol{r}^{(k)}$.

Theorem 4.7. *The Calculate_r algorithm (Algorithm 3) converges to a stable state if the following condition is satisfied:*

$$\frac{\phi_{il}}{\phi_{ij}\mu_j} + \frac{\phi_{il}}{\phi_{ij}r_j} - \frac{1}{\mu_l} - \frac{1}{r_l} \geq 0, \tag{50}$$

for all $i \in \mathcal{N}$, $j, l \in \mathcal{M}$, and $j \neq l$.

Proof. Let

$$g_{ij}(r_j, r_l) = \frac{\phi_{il}}{\phi_{ij}\mu_j} + \frac{\phi_{il}}{\phi_{ij}r_j} - \frac{1}{\mu_l} - \frac{1}{r_l}.$$

Then, we have

$$f_{ij}(\boldsymbol{r}) = \phi_{ij}\Lambda_i R_{i,\max} - \left(\frac{1}{\mu_j} + \frac{1}{r_j}\right) - \phi_{ij}\sum_{l=1, l\neq j}^{m}\left[g_{ij}\left(r_j, r_l\right)\left(\frac{r_l^2}{\mu_l} + r_l\right)\right].$$

If $g(r_j, r_l) \geq 0$, then it is easy to observe that $f_{ij}(\boldsymbol{r})$ decreases with the increase of r_l. Since $h_{ij}(\boldsymbol{r})$ (see Equation 49) increases with the increase of r_l, we can conclude that $F_j(\boldsymbol{r})$ (see Equation (47)) increases with the increase of r_l.

On the other hand, since $f_{ij}(\boldsymbol{r})$ increases with the increase of r_j and $g_{ij}(\boldsymbol{r})$ decreases with the increase of r_j, $F_j(\boldsymbol{r})$ decreases with the increase of r_j.

Hence, $\forall l \in \mathcal{M}$, if r_l increases, the value of $F_j(\boldsymbol{r})$ increases. To maintain the equation $F_j(\boldsymbol{r}) = 0$, r_j also increases. In addition, all the values in $\boldsymbol{r}$ are bounded. Therefore, we conclude the result in Theorem 4.7. This completes the proof and the result follows. □

Table 2. System Configurations

System parameters	Varied range
The number of cores for each server (c_j)	{2, 4, 8, 12}
Processing capacity of each core (μ_j)	[220, 480]
Request arrival rates of each user (Λ_i)	[10, 240]
Other parameters (ϵ, n, m)	(10^{-5}, 100, 300)

Table 3. Performance Requirements (Data I)

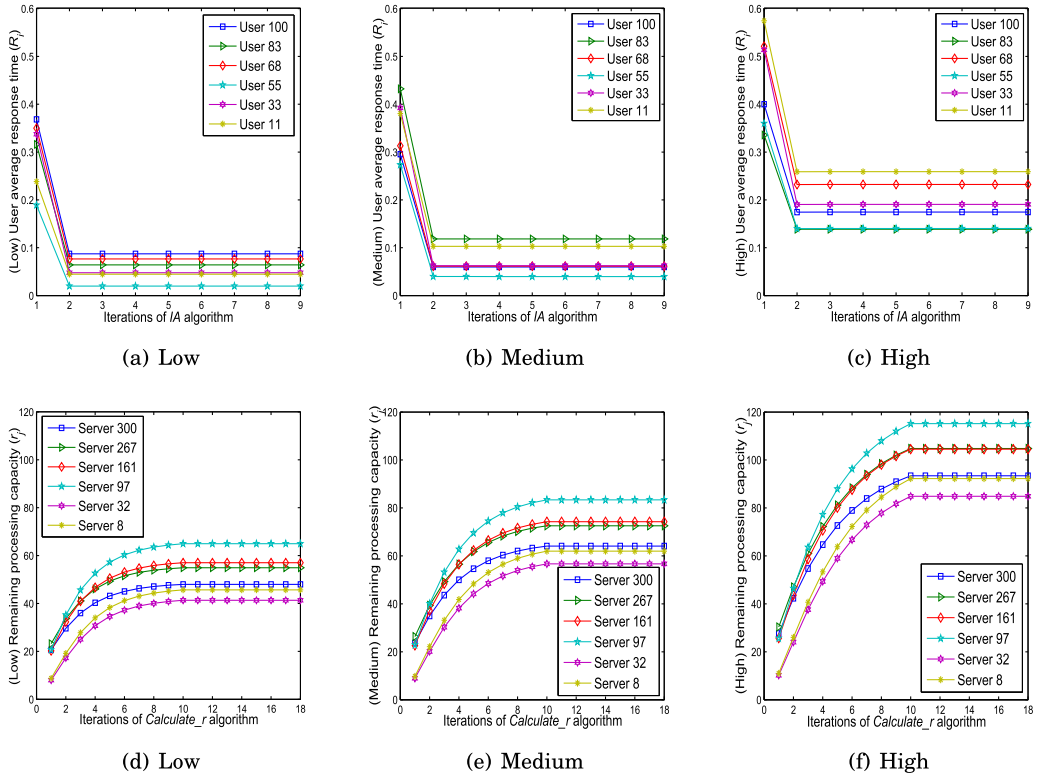
Performance requirements	Low	Medium	High
Performance requirements ($R_{i,\max}$)	[150, 300]	[80, 120]	[5, 50]

5 PERFORMANCE EVALUATION

In this section, we provide some numerical results to validate our analyses and illustrate the feasibility and effectiveness of our proposed $\mathcal{IA}$ algorithm.

In the simulation environment, we consider a cloud provider consisting of 300 servers. Each server has multiple identical cores. As shown in Table 2, the number of cores for each server is randomly and uniformly chosen from 2, 4, 8, and 12, and the processing rate (μ_j) of a core is randomly and uniformly chosen from 220 to 480. We assume that the request arrival rate (Λ_i) of each cloud user can vary from 10 to 240, and we randomly and uniformly select a value from the interval for each cloud user. The number of cloud users n is set as a constant 100, and ϵ is set as 10^{-5}. In Table 3, we show the varied ranges of average response time requirements ($R_{i,\max}$) of all cloud users. Specifically, we choose the average response time requirement values with three levels, i.e., the low quality level, medium quality level, and high quality level. Since the smaller value means higher quality requirement, the requirement values are varied from 150 to 300 with low quality level, from 80 to 120 with medium quality level, and from 5 to 50 with high quality level, respectively.

Figures 2(a) and 2(d) show an instance with low performance requirements. Specifically, Figure 2(a) presents the average response time results of six randomly selected cloud users (users 11, 33, 55, 68, 83, and 100) with the performance requirements of all users randomly and uniformly chosen from low quality service interval. We can observe that the average response times of all users seem to decrease with the increase of iteration number and finally reach a relatively stable state. That is, the average response times of all cloud users remain unchanged, i.e., reach a NBS after several iterations. It can also be seen that the developed algorithm converges to a NBS very quickly. Specifically, the average response time of each user has already achieved a relatively stable state after two iterations. Notice that, in each iteration of the $\mathcal{IA}$ algorithm, we use another subalgorithm ($Calculate\_\boldsymbol{r}$) to calculate the remaining processing capacities of all servers first, which also has an iterative process. Hence, the efficiency of the $\mathcal{IA}$ algorithm is also impacted by the efficiency of the $Calculate\_\boldsymbol{r}$ algorithm. In view of this, in Figure 2(d), we show the instance for the convergency of the $Calculate\_\boldsymbol{r}$ algorithm. We can observe that the remaining processing capacities of all servers tend to increase with the increase of iteration number and also reach a relative stable state. Namely, the aggregated requests on all servers are kept unchanged, which verifies the validness of Theorem 4.7. Furthermore, it can also be seen that the $Calculate\_\boldsymbol{r}$ algorithm converges very quickly. Specifically, the remaining processing capacities of all servers have already achieved a relatively stable state after 13 iterations. Therefore, our proposed $\mathcal{IA}$ algorithm can find a NBS in around 26 total iterations, which is very efficient.



(a) Low (b) Medium (c) High

(d) Low (e) Medium (f) High

Fig. 2. Convergency illustrations with Data I.

Similar to Figure 2(a) and (d), we show the results with medium performance requirements in Figure 2(b) and (e), and the results with high performance requirements in Figure 2(c) and (f).

From Figure 2(a)–(c), we can observe that the quality level of performance requirements has little impact on the efficiency of our proposed $\mathcal{IA}$ algorithm, i.e., no matter what the performance requirements of all cloud users are, the $\mathcal{IA}$ algorithm converges to a NBS very quickly (in around two iterations). However, the performance requirement levels impact the average response times of cloud users. Specifically, when the increase of performance requirement level is not very large (from *low* level to *medium* level), the average response times of some users (users 83, 55, 33, and 11) tend to increase while the average response times of some users (users 100 and 68) tend to decrease. On the other hand, when the performance requirements of cloud users are high enough, most of the cloud users tend to increase (see Figure 2(c)).

Figure 2(d)–(f) show the convergency of our proposed $Calculate\_\boldsymbol{r}$ algorithm. Since it is called by our proposed $\mathcal{IA}$ algorithm to calculate the remaining processing capacities of all servers in each iteration, it significantly impacts the efficiency of the $\mathcal{IA}$ algorithm. We can observe that the quality level of performance requirements of all cloud users also have little impact on the efficiency of the $Calculate\_\boldsymbol{r}$ algorithm. That is, no matter what the performance requirements are, the $Calculate\_\boldsymbol{r}$ algorithm converges to a relatively stable state very quickly (in around 13 iterations). However, on the other hand, the performance requirement levels impact the remaining processing capacities (the aggregated requests) of all servers. Specifically, the processing capacities of most servers tend to increase. This means that the aggregated requests on some servers tend to

Table 4. Performance Requirements (Data II)

Performance requirements	Low	Medium	High
Performance requirements ($R_{i,\max}$)	[0.2, 1.0]	[0.2, 0.6]	[0.2, 0.4]

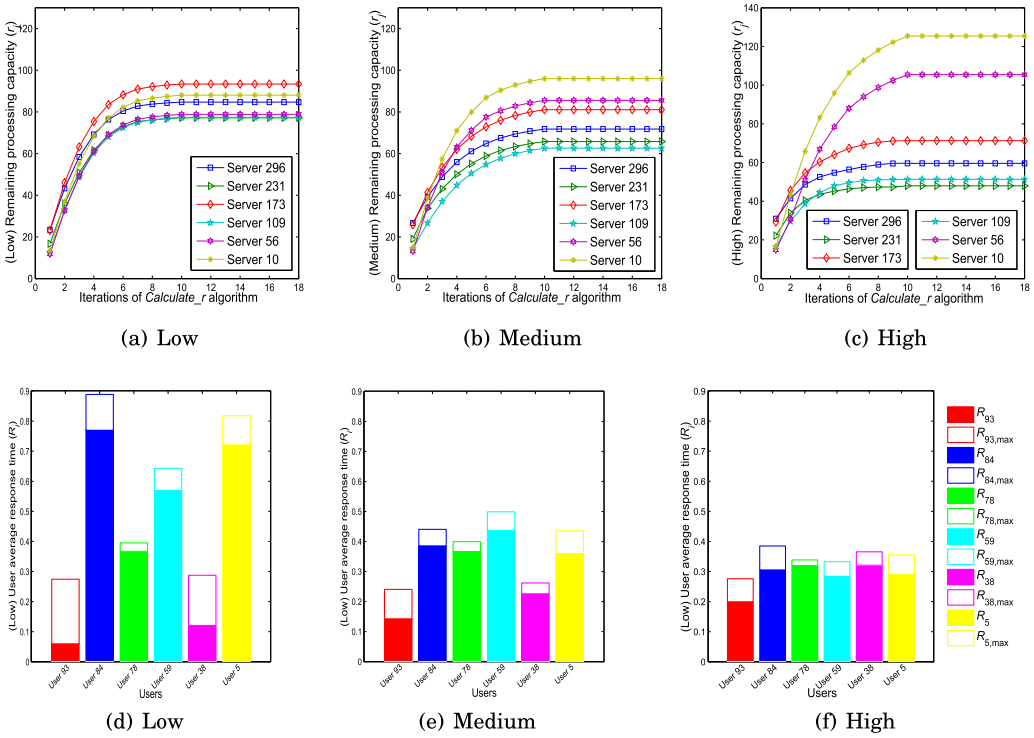


Fig. 3. Performance illustrations with Data II.

decrease with the increase of performance requirement level. The reason behind this lies in that with the increase of performance requirement level, the performance requirements of all cloud users cannot be guaranteed by just using faster servers, which can also partly validate the results shown in Figure 2(a)–(c).

Notice that, as shown in Table 3, even the interval with the high performance requirement is also from 5 to 50, while the final reached average response times of the randomly selected users are all less than 1 (see Figure 2(a)–(c)). That is to say, the average response requirements are far greater than the actual reached average response times. Hence, to further investigate our scheme, we conduct experiments on other tighter parameter configurations, which are shown in Table 4. The performance requirement values of users are varied from 0.2 to 1.0 with low quality level, from 0.2 to 0.6 with medium quality level, and from 0.2 to 0.4 with high quality level, respectively.

Figure 3(a)–(c) show the remaining processing capacities (r_i) of servers with the increase of iterations. Specifically, we randomly and uniformly select some servers (servers 296, 231, 173, 109, 56, and 10) and track their remaining processing capacities. We can observe that similar to Figure 2(d)–(f), the *Calculate_r* algorithm converges to a relatively stable state very quickly (around 13 iterations), which shows the feasibility and stability of our proposed *Calculate_r* algorithm. In addition, with the increase of the performance requirement level (from *low* to *medium*

to *high*), the final remaining processing capacities of some servers (servers 56 and 10) tend to increase, while those of some other servers (servers 296, 231, 173, and 109) tend to decrease. The reason behind this lies in that with the increase of performance requirement level, to minimize the average response times of users, some slower servers tend to migrate their requests to faster servers. This results in the increase of requests on some servers (servers 296, 231, 173, and 109) and decrease on some other servers (servers 56 and 10).

Figure 3(d)–(f) show the actual average response times of users and their corresponding performance requirements. We can observe that with the increase of the performance requirement level, the actual average response times of some users (users 84, 78, 59, and 5) decrease, while those of some other users increase (users 93 and 38). The reason behind this lies in that with the increase of performance requirement level, some requests on slower servers have to be migrated to faster servers to decrease the average response times of some users. Obviously, these migrations can also result in an increase of the average response times of some other users. We can also observe that the obtained average response times of all users are less than their corresponding performance requirements, which caters to our scheduling problem.

6 CONCLUSIONS

With the popularization of cloud computing and its many advantages such as cost-effectiveness, flexibility, and scalability, more and more applications are moved from local to cloud. However, most cloud providers cannot make use of the limited resources and guarantee performances of all cloud users simultaneously. We do this work to find a scheduling scheme which simultaneously guarantees the average response time requirements of all users, although we only consider an average response time metric.

Each cloud user submits requests with an average response time requirement. The cloud provider tries to find a scheduling scheme, i.e., allocating requests to multiple servers, such that the requirements of all cloud users can be guaranteed. We formulate the scheduling problem into cooperative game and try to find a NBS which can satisfy the average response time demands of all users at the same time. We first prove the existence of NBS and then analyze its computation. For the situation when all allocating substreams are strictly positive, we propose a $\mathcal{CA}$, which can find the NBS very efficiently. For the general case, we propose an $\mathcal{IA}$ which is based on duality theory. The convergence of the $\mathcal{IA}$ algorithm is also analyzed. Finally, we conduct some numerical calculations. The results show the feasibility and effectiveness of our proposed $\mathcal{IA}$ algorithm.

As part of future directions, we will dynamically configure the multiple servers in cloud and study the relationship between the cloud provider and multiple users. Another direction is to study the cloud choice among multiple different cloud providers or determine a proper mixed choosing strategy.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to the editor(s) and anonymous reviewers for their comments and suggestions, which have greatly helped to improve the quality of the manuscript.

REFERENCES

Shailendra S. Aote and M. U. Kharat. 2009. A game-theoretic model for dynamic load balancing in distributed systems. In *Proceedings of the International Conference on Advances in Computing, Communication and Control.* ACM, 235–238.

Jean-Pierre Aubin. 2007. *Mathematical Methods of Game and Economic Theory.* Courier Dover Publications.

Avrim Blum, Jamie Morgenstern, Ankit Sharma, and Adam Smith. 2015. Privacy-preserving public information for sequential games. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science.* ACM, 173–180.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press.

Junwei Cao, Kai Hwang, Keqin Li, and Albert Y. Zomaya. 2013. Optimal multiserver configuration for profit maximization in cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 24, 6 (2013), 1087–1096.

Junwei Cao, Keqin Li, and Ivan Stojmenovic. 2014. Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers. *IEEE Transactions on Computers* 63, 1 (2014), 45–58.

Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. 2012. Optimization of resource provisioning cost in cloud computing. *IEEE Transactions on Services Computing* 5, 2 (2012), 164–177.

Bin Dong, Xiuqiao Li, Qimeng Wu, Limin Xiao, and Li Ruan. 2012. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers. *Journal of Parallel and Distribed Computing* 72, 10 (2012), 1254–1268.

Amos Fiat, Anna Karlin, Elias Koutsoupias, and Angelina Vidali. 2013. Approaching utopia: Strong truthfulness and externality-resistant mechanisms. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ACM, 221–230.

Y. Gao, Y. Chen, and K. J. R. Liu. 2012. Cooperation stimulation for multiuser cooperative communications using indirect reciprocity game. *IEEE Transactions on Communications* 60, 12 (2012), 3650–3661.

Davide Grossi and Paolo Turrini. 2010. Dependence theory via game theory. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 1147–1154.

Jian Guo, Fangming Liu, Dan Zeng, John Lui, and Hai Jin. 2013. A cooperative game based allocation for sharing data center networks. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2139–2147.

Dong-Ki Kang, Seong-Hwan Kim, Chan-Hyun Youn, and Min Chen. 2014a. Cost adaptive workflow scheduling in cloud computing. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*. ACM, 65.

Seungmin Kang, Bharadwaj Veeravalli, and Khin Mi Mi Aung. 2014b. Scheduling multiple divisible loads in a multi-cloud system. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC'14)*. IEEE, 371–378.

Hamzeh Khazaei, Jelena Misic, and Vojislav Misic. 2013. A fine-grained performance model of cloud computing centers. *IEEE Transactions on Parallel and Distributed Systems* 24, 11 (2013), 2138–2147.

Cinar Kilcioglu and Justin M Rao. 2016. Competition on price and quality in cloud computing. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1123–1132.

Jörn Künsemöller and Holger Karl. 2012. A game-theoretical approach to the benefits of cloud computing. In *Economics of Grids, Clouds, Systems, and Services*. Springer, 148–160.

Keqin Li. 2013. Optimal load distribution for multiple heterogeneous blade servers in a cloud computing environment. *Journal of Grid Computing* 11, 1 (2013), 27–46.

Kenli Li, Chubo Liu, and Keqin Li. 2014. An approximation algorithm based on game theory for scheduling simple linear deteriorating jobs. *Theoretical Computer Science* 543, 0 (2014), 46–51.

Na Li and J. R. Marden. 2011. Designing games for distributed optimization. In *Proceedings of the 2011 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC'11)*. 2434–2440.

C. Liu, K. Li, C. Xu, and K. Li. 2016. Strategy configurations of multiple users competition for cloud service reservation. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 508–520.

Dantong Liu, Yue Chen, Kok Keong Chai, and Tiankui Zhang. 2014. Nash bargaining solution based user association optimization in HetNets. In *Proceedings of the 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC'14)*. IEEE, 587–592.

Maggie Mashaly and Paul J. Kühn. 2012. Load balancing in cloud-based content delivery networks using adaptive server activation/deactivation. In *Proceedings of the 2012 International Conference on Engineering and Technology (ICET'12)*. IEEE, 1–6.

G. Mc Evoy and Bruno Schulze. 2011. Understanding scheduling implications for scientific applications in clouds. In *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*. ACM, 3.

J. Mei, K. Li, A. Ouyang, and K. Li. 2015. A profit maximization scheme with guaranteed quality of service in cloud computing. *IEEE Transactions on Computers* PP, 99 (2015), 1–1.

M. R. Musku, A. T. Chronopoulos, D. C. Popescu, and A. Stefanescu. 2010. A game-theoretic approach to joint rate and power control for uplink CDMA communications. *IEEE Transactions on Communications* 58, 3 (2010), 923–932.

Martin J. Osborne and Ariel Rubinstein. 1994. *A Course in Game Theory*. MIT Press.

Satish Penmatsa and Anthony T. Chronopoulos. 2011. Game-theoretic static load balancing for distributed systems. *Journal of Parallel and Distributed Computing* 71, 4 (2011), 537–555.

M. M. Rana, Saurabh Bilgaiyan, and Utsav Kar. 2014. A study on load balancing in cloud computing environment using evolutionary and swarm based algorithms. In *Proceedings of the 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT'14)*. IEEE, 245–250.

Nageswara S. V. Rao, Stephen W. Poole, Fei He, Jun Zhuang, Chris Y. T. Ma, and David K. Y. Yau. 2012. Cloud computing infrastructure robustness: A game theory approach. In *Proceedings of the 2012 International Conference on Computing, Networking and Communications (ICNC'12)*. IEEE, 34–38.

M. Elena Renda, Giovanni Resta, and Paolo Santi. 2012. Load balancing hashing in geographic hash tables. *IEEE Transactions on Parallel and Distributed Systems* 23, 8 (2012), 1508–1519.

G. Scutari, D. P. Palomar, F. Facchinei, and Jong-Shi Pang. 2010. Convex optimization, game theory, and variational inequality theory. *IEEE Signal Processing Magazine* 27, 3 (May 2010), 35–49.

Gesualdo Scutari, Daniel P. Palomar, Francisco Facchinei, and Jong-Shi Pang. 2012. Monotone games for cognitive radio systems. In *Distributed Decision Making and Control*, Rolf Johansson and Anders Rantzer (Eds.). Lecture Notes in Control and Information Sciences, Vol. 417. Springer, London, 83–112.

Aarti Singh, Dimple Juneja, and Manisha Malhotra. 2015. Autonomous agent based load balancing algorithm in cloud computing. *Procedia Computer Science* 45 (2015), 832–841.

R. Subrata, A. Y. Zomaya, and B. Landfeldt. 2008. A cooperative game framework for QoS guided job allocation schemes in grids. *IEEE Transactions on Computers* 57, 10 (2008), 1413–1422.

David R. M. Thompson and Kevin Leyton-Brown. 2013. Revenue optimization in the generalized second-price auction. In *Proceedings of the 14th ACM Conference on Electronic Commerce*. ACM, 837–852.

Cong Xu, Sahan Gamage, Pawan N. Rao, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan Xu. 2012. vSlicer: Latency-aware virtual machine scheduling via differentiated-frequency CPU slicing. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 3–14.

Xin Xu and Huiqun Yu. 2014. A game theory approach to fair and efficient resource allocation in cloud computing. *Mathematical Problems in Engineering* 2014 (2014).

Haïkel Yaïche, Ravi R. Mazumdar, and Catherine Rosenberg. 2000. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions on Networking (TON)* 8, 5 (2000), 667–678.

Received May 2016; revised May 2017; accepted September 2017