J Supercomput (2010) 54: 252–269
DOI 10.1007/s11227-009-0318-1

A game-theoretic method of fair resource allocation for cloud computing services

Guiyi Wei · Athanasios V. Vasilakos · Yao Zheng · Naixue Xiong

Published online: 29 July 2009

Abstract As cloud-based services become more numerous and dynamic, resource provisioning becomes more and more challenging. A QoS constrained resource allocation problem is considered in this paper, in which service demanders intend to solve sophisticated parallel computing problem by requesting the usage of resources across a cloud-based network, and a cost of each computational service depends on the amount of computation. Game theory is used to solve the problem of resource allocation. A practical approximated solution with the following two steps is proposed. First, each participant solves its optimal problem independently, without consideration of the multiplexing of resource assignments. A Binary Integer Programming method is proposed to solve the independent optimization. Second, an evolutionary mechanism is designed, which changes multiplexed strategies of the initial optimal solutions of different participants with minimizing their efficiency losses. The algorithms in the evolutionary mechanism take both optimization and fairness into account. It is demonstrated that Nash equilibrium always exists if the resource allocation game has feasible solutions.

G. Wei
Zhejiang Gongshang University, Hangzhou, Zhejiang, People's Republic of China
e-mail: weigy@mail.zjgsu.edu.cn

A.V. Vasilakos (✉)
National Technical University of Athens, Athens, Greece
e-mail: vasilako@ath.forthnet.gr

Y. Zheng
Zhejiang University, Hangzhou, Zhejiang, People's Republic of China
e-mail: yao.zheng@zju.edu.cn

N. Xiong
Department of Computer Science, Georgia State University, Atlanta, USA
e-mail: nxiong@cs.gsu.edu

Keywords Cloud computing · Game theory · Resource allocation

1 Introduction

Cloud computing becomes more and more popular in large-scale computing and data store recently due to it enables the sharing of computing resources that are distributed all over the world. Cloud computing infrastructures break down the physical barriers inherent in isolated systems, automate the management of the group of resources as a single entity, and provide computational power and data storage facilities to users, ranging from a user accessing a single laptop to the allocation of thousands of computing nodes distributed around the world.

Cloud computing is a natural evolution for data and computation centers with automated systems management, workload balancing, and virtualization technologies. Cloud-based services integrate globally distributed resources into seamless computing platforms. Recently, a great deal of applications are increasingly focusing on third-party resources hosted across the Internet and each has varying capacity.

In the past few years, many extant systems have the notion that resource, whose information was hidden by from the user through virtualization, can be acquired and released on-demand. However, most cloud computing systems in operation are proprietary, and rely upon infrastructure that is invisible to the research community, or explicitly designed not to be instrumented and modified by systems researchers interested in cloud computing systems.

It is known that the underlying cloud computing environment is inherently large-scale, complex, heterogeneous and dynamic, globally aggregating large numbers of independent computing and communication resources and data stores. In an open cloud computing framework, scheduling tasks with guaranteeing QoS constrains present a challenging technical problem.

In large scale, how can resource manager in decentralized cloud computing networks efficiently allocate scarce resources among competing interests? Constraints are used by the inherent architecture of cloud computing systems. These constraints are counterbalanced by the requirement to design mechanisms that efficiently allocate resources, even when the system is being used by participants who have only their own interests. For example, users want to get hotel information in a city by an integrated web search engine. Performing this task includes simultaneous invoking web page search service, map search service, and semantic computing services. Provisioning of these services must coordinate completion time of these subtasks to provide unified response to clients. In the background, service provider schedules the correlated services and allocates different resources for them. Another example, while users solve scientific computing problems using cloud resources, the task is divided into several parallel subtasks which are computation-intensive and frequently communicate with each other. Provisioning of this class computing services needs efficient resources management system and effective scheduling algorithms.

As cloud-based services become more numerous and dynamic, resource provisioning becomes more challenging. A QoS-constrained resource allocation problem is considered in this paper, where service demanders intend to solve sophisticated computing problem by requesting the usage of resources across a cloud-based network, and a cost of each network node depends on the amount of computation. Using cloud computing platforms to perform QoS-constrained and computation-intensive tasks is studied. This paper is also focused on the parallel tasks allocation problem on unrelated machines connected across the Internet. The scheduling problem is one of the most fundamental challenges in most real cloud-based applications.

The main contributions of the paper include: (1) proposal of a deadline and budget constrained cost-time optimization algorithm for scheduling dependent subtasks with considerations of communications between them; (2) a design for an evolutionary mechanism, which changes multiplexed strategies of the initial optimal solutions of different participants with minimizing their efficiency losses; and (3) demonstration of Nash equilibrium in existence in the resource allocation game with the specialized settings.

2 Related works

Cloud computing systems fundamentally provide access to large amounts of data and computational resources that can be acquired and released on-demand. As consumers rely on cloud providers to supply more of their computing needs, they will require specific QoS to be maintained by their providers in order to meet their requirements. In fact, cloud computing systems provide best-effort collaborative computation service, which is not sufficient for solving engineering and scientific problem in several aspects: quality of service is not guaranteed and a bounded completion time cannot be offered.

Various resource management methods with different policies and principles were published in some papers [1, 2, 6, 9, 14, 17, 19, 24, 28, 30, 32].

Most of the scheduling algorithms assume the tasks are independent of each other. Under this assumption, the existing algorithms can still work with many loose-coupling service-integrated applications. However, the majority of complex cloud-based applications consist of multiple subtasks and require communications among tasks. The collaboration must be taken into account when allocating resources to them. Without considering the communications among tasks, algorithms are obviously limited, and cannot take full advantage of the powerful cloud computing. Therefore, it is very challenging to schedule general dependent tasks.

The dependent task scheduling problem is NP-hard in its general form [11, 16, 23]. Many heuristic algorithms have been proposed for near-optimal solutions [3, 7, 12, 18, 20, 21, 29, 31]. Game theory studies the problems in which players maximize their returns which depend also on actions of other players. Numerous studies [4, 5, 8, 10, 11, 13, 15, 22, 25] have proposed game-theoretic method to solve the optimization problem of resource allocation in network systems from the viewpoint of resource owners. However, there is still lack of practicable solution for cloud computing systems because most cloud-based computational services are multiple QoS-constrained. Therefore, for the first step, the authors try to add some constraints on communications in order to achieve an improved scheduling algorithm. This paper presents a new game-theoretic method to schedule dependent tasks with time and cost constraints. An evolutionary mechanism is designed to fairly and approximately solve the NP-hard problem.

3 Problem modeling

Suppose that n tasks (users) share m computational resources. Each task S_i consists of $k(i)$ parallel and dependent subtasks with equal amount of computation. Each resource R_j has a fixed price p_j according to its capacity. When multiple subtasks are assigned to a resource R_j, they proportionally share R_j's capacity and expense. The goal is to assign each subtask to a specific resource in order to minimize the total "cost"—the expense and the time for completing all these n tasks. Note that the subtasks in a task would sometimes communicate with each other, which will make the problem even more complicated.

A solution of the scheduling problem is a non-negative matrix a of n rows, one for each task, and m columns, one for each resource. The entry a_{ij} is the amount of subtasks of the task S_i allocated to resource R_j. Let a_i represent the ith row of matrix a. Then allocation vector a_i satisfies $\sum_{a_{ij} \in a_i} a_{ij} = k(i)$. Derived from matrix a, another two $n \times m$ matrixes are obtained: completion time matrix T and expense matrix E. The entry t_{ij} of T is the turnaround time it takes for resource R_j to complete a_{ij} subtasks of the task S_i. Because all subtasks of S_i are parallel and dependent, the completion time of task S_i is $\max\{t_{ij} \mid t_{ij} \in t_i\}$, where t_i denotes the vector of the ith row of matrix T. The entry e_{ij} of matrix E is the expense S_i pays for resource R_j to complete a_{ij} subtasks. So, the expense of task S_i is $\sum_{j=1}^{m} e_{ij}$. In general, there is a trade-off between completion time and expense for each task. Assume that all participants have the same views of value towards monetary expense and time. Let w_t and w_e denote the weights of completion time and expense, respectively. Therefore, the total "cost" of task S_i is $w_t \cdot \max_{t_{ij} \in t_i} \{t_{ij}\} + w_e \cdot \sum_j e_{ij}$. Let

$$u_i(a_i) = \frac{1}{w_t \cdot \max_{t_{ij} \in t_i} \{t_{ij}\} + w_e \cdot \sum_j e_{ij}}$$

denote the utility of task S_i. The objective of each task is to maximize its utility.

Assume that: (1) the capacity of each resource is inelastic and undividable; (2) each resource can be allocated to multiple subtasks of different tasks; and (3) all resources use time-sharing policy to schedule tasks in the operating system level. Let $\hat{t}_{ij}$ denote the execution time it takes for resource R_j solely to complete one subtask of task S_i without consideration of communication time. It can be inferred that $t_{ij} = \sum_j a_{ij} \cdot \hat{t}_{ij}$ and $e_{ij} = a_{ij} \cdot \hat{t}_{ij} \cdot p_j$ when only one subtask assigns to one resource. Without loss of generality, this paper assumes that the price vector of all resources $p = (p_1, p_2, .., p_m)$ satisfies $p_1 < p_2 < \cdots < p_m$, and the corresponding execution time of any subtask of an arbitrary task S_i satisfies $\hat{t}_{i1} > \hat{t}_{i2} > \cdots > \hat{t}_{im}$.

To illustrate the problem model, a use case is designed as the following. (1) There are five computational resources ($R_1 - R_5$), where the price vector $p = (1, 1.2, 1.5, 1.8, 2)$. And (2) there are three tasks (S_1, S_2, S_3), where S_1 has two subtasks, S_2 has three subtasks and S_3 has four subtasks. The available resources and tasks are listed below.

R_1:	One node of a supercomputer (SGI Onyx 3900).
R_2:	One node of a Cluster (TC 4000L).
R_3:	A workstation (IBM xSeries 255 8685-B1X).
R_4:	A desktop PC (Intel Core 2 Duo, 3.0 GHz).
R_5:	A desktop PC (Intel Core 2 Duo, 2.66 GHz).
S_1:	A parallel computational fluid dynamics program.
S_2:	A parallel computational solid mechanics program.
S_3:	A parallel Monte Carlo computational program.

The execution time matrix

$$\hat{t}_{ij} = \begin{pmatrix} 6 & 5 & 4 & 3.5 & 3 \\ 5 & 4.2 & 3.6 & 3 & 2.8 \\ 4 & 3.5 & 3.2 & 2.8 & 2.4 \end{pmatrix}.$$

Assume S_1 chooses $\{R_1, R_2\}$, S_2 chooses $\{R_1, R_2, R_3\}$ and S_3 chooses $\{R_2, R_3, R_4, R_5\}$, i.e., the strategies of S_1, S_2 and S_3 are $a_1 = (1,1,0,0,0)$, $a_2 = (1,1,1,0,0)$ and $a_3 = (0,1,1,1,1)$ respectively. Then allocation

$$a = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Since all tasks proportionally share the capacity and expense of allocated resources, according to this allocation, the final execution time matrix

$$t_{ij} = \begin{pmatrix} 12 & 15 & 0 & 0 & 0 \\ 10 & 12.6 & 7.2 & 0 & 0 \\ 0 & 10.5 & 6.4 & 2.8 & 2.4 \end{pmatrix}$$

and the expense matrix

$$\begin{aligned} e_{ij} &= \begin{pmatrix} 12 \times \frac{p_1}{2} & 15 \times \frac{p_2}{3} & 0 & 0 & 0 \\ 10 \times \frac{p_1}{2} & 12.6 \times \frac{p_2}{3} & 7.2 \times \frac{p_3}{2} & 0 & 0 \\ 0 & 10.5 \times \frac{p_2}{3} & 6.4 \times \frac{p_3}{2} & 2.8 \times p_4 & 2.4 \times p_5 \end{pmatrix} \\ &= \begin{pmatrix} 6 & 6 & 0 & 0 & 0 \\ 5 & 5.04 & 5.4 & 0 & 0 \\ 0 & 4.2 & 4.8 & 5.04 & 4.8 \end{pmatrix}. \end{aligned}$$

To simplify the computation, assume $w_t = w_e = 0.5$; then

$$u_1(a_1) = \frac{1}{0.5 \times (15 + (6+6))} \simeq 0.0741,$$

$$u_2(a_2) = \frac{1}{0.5 \times (12.6 + (5 + 5.04 + 5.4))} \simeq 0.0713,$$

$$u_3(a_3) = \frac{1}{0.5 \times (10.5 + (4.2 + 4.8 + 5.04 + 4.8))} \simeq 0.0682.$$

Let each task be a market participant. a_i is the strategy of participant S_i. Then allocation matrix $a = (a_1, a_2, \dots, a_n)^T$ is the strategy set of all participants. The standard game-theoretic convention a_{-i} is used to denote what remains from a when its ith element a_i is dropped; similarly, (a'_i, a_{-i}) denotes the allocation after replacing a_i by a'.

It is considered that the design of market mechanisms is for such settings which are robust to gaming behavior by market participants. Given complete knowledge of the system, it would be natural for each participant to try to solve the following optimization problem:

$$\text{Maximize} \quad u_i(a_i) \tag{1}$$

$$\text{Subject to} \quad \sum_{a_{ij} \in a_i} a_{ij} = k(i), \tag{2}$$

$$a_{ij} \geq 0. \tag{3}$$

Unfortunately, it is NP-complete to find optimal allocations for the resource allocation problem. However, approximated solution with the two following steps is effective. That Nash equilibrium of the resource allocation game always exists is demonstrated in the following two steps.

(1) Each participant solves his optimal problem independently and solely without considering the multiplexing of resources, i.e., assuming $t_{ij} = \hat{t}_{ij}$ for all $i \in [1..n]$ and $j \in [1..m]$ (Sect. 4).
(2) An evolutionary mechanism is defined, which changes multiplexed strategies of the initial optimal solutions of different participants with minimizing their efficiency losses (Sect. 5).

4 Independent optimization

4.1 Assumptions

(1) For each single task, while it is assigned to a specific resource in the cloud, the time spent on it can be known. Many techniques are available to achieve this [9, 14].
(2) Subtasks are dependent, and communicate with each other. Bandwidth cost in communications is not taken into consideration. The communication happens whenever a certain percentage amount of work of every subtask has been completed for all the tasks. For example, the communication happens when 10% of work of every subtask has been done, and then, communication happens again when another 6% work of every subtask has been done, and so on. Before execution, the value of percentage need not be known, but the times of communication that happened as well as the scale of communications must be known.

    Although this is a constraint, most problems in scientific computing services and parallel data analysis services satisfy this assumption. For example, computations in computational structural analysis, in computational fluid dynamics, and in DNA sequencing, all satisfy this assumption. Also, a cloud-based service with independent subtasks can be considered as the service with subtasks that communicate after 100% of work has been completed. Therefore, the algorithm works with both independent and dependent cloud-based services.

(3) According to most accounting systems, the charge for a unit of time is proportional to the resource used by the user at that moment. Provided that, for a specific processor, if its resource used is very close to zero in a period of time, the cost it charges is very close to zero. Thus, it is reasonable to assume that when waiting for communication, no money is consumed.

4.2 Algorithm description

4.2.1 Definition of the object function

A proper function to measure the "cost" has been defined. It is good to define the function as a weighed sum of the expense and the whole completion time [19]. For example, participants consider one unit of time as valuable as one hundred units of money, then set $w_m : w_t = 1 : 1$, and $w_m + w_t = 1$. The participants (or tasks) can also set the deadline (denoted as T_0) and the maximal expense that can be afforded (denoted as M_0). Independent optimization algorithm only concerns one task which contains multiple subtasks. Let b denote the allocation matrix which consists of k rows, one for each subtask, and m columns, one for each resource. Based on this logic, the object function is defined as (4):

$$\min Z = w_m \cdot \left(\sum_{i=1}^{k} \sum_{j=1}^{m} p_j \cdot \hat{t}_{ij} \cdot b_{ij} \right) + w_t \cdot T_{\text{turnaround}}, \tag{4}$$

where

$$T_{\text{turnaround}} = \sum_{l=1}^{q} t_l + \max_{i,j} \left\{ (b_{ij} \cdot \hat{t}_{ij}) \cdot \left(1 - \sum_{l=1}^{q} n_l \right) \right\}, \tag{5}$$

and

$$t_l = \max_{i,j} \{ b_{ij} \cdot \hat{t}_{ij} \cdot n_l \} + \max_{i,j} \{ b_{ij} \cdot t_{ij}^l \}, \quad l \in [1..q]. \tag{6}$$

Here, q stands for the times of communications that happened during the execution of the task.

Equation (6) gives the value of the duration from the end of the $(l-1)$th communication to the end of the lth communication, and n_l is the percentage amount of work completed during these two communications. For each l, t_{ij}^l stands for the time for the lth communication for subtask i assigned to resource R_j (the value of b_{ij} determines whether or not subtask i is assigned to R_j). Equation (6) means that while another n_l (percentage amount) of every subtask is finished, the lth communication will happen. The reason for using the first "max" is that the communication will not happen until all the tasks are ready for communication. $\max_{i,j}\{b_{ij} \cdot t_{ij}^l\}$ stands for the time spent on the lth communication. Here, "max" is used because the quality of network varies from place to place, and it is the slowest network that determines the time for communications.

Equation (5) has given the value of the task completion time of k subtasks. The term $(1-\sum_{l=1}^{q} n_l)$ is used because after the last communication, $(1-\sum_{l=1}^{q} n_l)$ of each subtask is left.

4.2.2 Constrained conditions

For each pair of i and j, as in the above assumption, $\hat{t}_{ij}$ is known. The partition of the allocation rule is described using indicator values $a_{ij} \in \{0, 1\} : a_{ij} = 1$ iff subtask i is allocated to resource j. Of course, each task is allocated to exactly one machine, or more formally, $\sum_{j=1}^{m} a_{ij} = 1$. The constrained conditions of the optimal problem are described as (7)–(11):

$$b_{ij} = 0 \text{ or } 1, \quad \text{for all } i \in [1..k] \text{ and } j \in [1..m], \tag{7}$$

$$T_{\text{turnaround}} \leq T_0, \tag{8}$$

$$\sum_{i=1}^{k} \sum_{j=1}^{m} p_j \cdot \hat{t}_{ij} \cdot b_{ij} \leq M_0, \tag{9}$$

$$\sum_{j=1}^{m} b_{ij} = 1, \quad \text{for all } i \in [1..k], \tag{10}$$

$$\sum_{i=1}^{k} b_{ij} \leq 1, \quad \text{for all } j \in [1..m]. \tag{11}$$

Inequations (8) and (9) mean "deadline and budget constrained." Equation (10) means that each single task should be assigned to one and only one processor. Inequation (11) means that each single processor should process at most one of those tasks. That is, because communications cannot happen until each of those tasks has finished by the same percentage, which has been assumed in Sect. 4.1.

4.3 Problem simplification and solving

Combining (5) and (6), the following equation (12) is obtained:

$$T_{\text{turnaround}} = \max_{i,j} \{ b_{ij} \cdot \hat{t}_{ij} \} + \sum_{l=1}^{q} \left(\max_{i,j} \{ b_{ij} \cdot t_{ij}^{l} \} \right). \tag{12}$$

A large proportion of cloud-based computing services is spent most of the time on computing, and the time for communications is relatively disregarded. (It does not mean that communications can be ignored, because usually much time is spent on waiting before communications.) When the time transferring data is very small, the difference among the networks is ignored, and the term $\max_{i,j}\{b_{ij} \cdot t_{ij}^{l}\}$ is replaced by t_l. Then, (12) is replaced by the following equation (13):

$$T_{\text{turnaround}} = \max_{i,j}\{b_{ij} \cdot \hat{t}_{ij}\} + \sum_{l=1}^{q} t_l. \tag{13}$$

Now, (13) is replaced by the inequation (14) and it is considered a constrain condition. It can be inferred that the replacement will not change the solution. The proof is omitted here, due to the space limitation.

$$T_{\text{turnaround}} \geq (b_{ij} \cdot \hat{t}_{ij}) + \sum_{l=1}^{q} t_l, \quad \text{for all } i \in [1..k] \text{ and } j \in [1..m]. \tag{14}$$

Except that b_{ij} should be binary integers, all the constrained conditions in the model are linear. Thus, the model is straightforward, and it is reduced to a classical *Binary Integer Programming* problem; and a lot of methods can be used.

In fact, some tasks may have some other requirements, for example, reliability. In such cases, not all the resources in a cloud are suitable for the tasks. If a certain resource is not suitable for a certain task, set the corresponding term $\hat{t}_{ij}$ to be greater than T_0. In this way, the algorithm can avoid assigning that task to that resource. Therefore, the algorithm can schedule tasks with QoS requirements.

4.4 Example of problem solving

Based on the example designed in Sect. 3, for the three tasks, the independent optimization algorithm runs and gets the following results: $a_1 = (0,0,0,1,1)$, $a_2 = (0,0,1,1,1)$ and $a_3 = (1,1,1,0,1)$.

Then allocation matrix is

$$a = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

The expected utilities of the three tasks are $u_1(a_1) \simeq 0.0633$, $u_2(a_2) \simeq 0.0500$ and $u_3(a_3) \simeq 0.0459$, respectively. But, they cannot obtain their expected utilities because there are some resources allocated to more than one subtask. So, actual turnaround time of some multiplexed subtasks will be longer than their expected time.

According to the allocation matrix a, the actual execution time matrix is

$$t_{ij} = \begin{pmatrix} 0 & 0 & 0 & 7 & 9 \\ 0 & 0 & 7.2 & 6 & 8.4 \\ 4 & 3.5 & 6.4 & 0 & 7.2 \end{pmatrix}.$$

The expense matrix is

$$e_{ij} = \begin{pmatrix} 0 & 0 & 0 & 6.3 & 6 \\ 0 & 0 & 5.4 & 5.4 & 5.6 \\ 4 & 4.2 & 4.8 & 0 & 4.8 \end{pmatrix}.$$

The actual utilities of the three tasks are $u'_1(a_1) \simeq 0.0469$, $u'_2(a_2) \simeq 0.0403$ and $u'_3(a_3) \simeq 0.0400$, respectively. Obviously, $u'_i < u_i$. The strategies $a_1 = (0, 0, 0, 1, 1)$, $a_2 = (0, 0, 1, 1, 1)$ and $a_3 = (1, 1, 1, 0, 1)$ are only the initial strategies of the three tasks. As an evolutionary game, the strategy of each task is common knowledge to every participant. In the next section, evolutionary optimization algorithms are proposed to approximate the utility of each task to its optimal value.

5 Evolutionary optimization

In this game, no one knows what global allocation will be before each task chose its first strategy. As described in Sect. 4, they all choose the optimal strategy with ignoring others. The possible result of initial optimal solutions of all tasks includes the following two occasions: non-multiplexing and multiplexing. Assume the previous strategies of all participants are common knowledge. When some resources are multiplexed, they will begin the next round game based on the result of the first round. The subsequent strategy change must satisfy the necessary condition: the objective new strategy of any task must increase all tasks' utility or all other tasks can choose corresponding new strategies to increase their utilities. Additionally, the process of the evolution changes only one subtask assignment at one time and the change should be known to all participants.

In this section, an evolutionary mechanism is presented by introducing the equilibrium concept. The concept of Nash equilibrium is in some sense the analog of centralized optimal design in the context of multiple distributed selfish participants. The objective of evolutionary mechanism is to achieve final optimization by changing multiplexed strategies of the initial optimal solutions of different participants. The two possibilities (non-multiplexing and multiplexing) both leading to market equilibrium are to be demonstrated.

5.1 Non-multiplexing

Definition 1 (Non-multiplexing Allocation) Given that a resource allocation game has n tasks and m resources, each task consists of multiple same subtasks. If an $n \times m$ matrix a satisfies the condition $\forall a_{ij} : (a_{ij} \leq 1) \wedge (\sum_{i=1}^{n} a_{ij} \leq 1)$, a is a *Non-multiplexing Allocation.*

Proposition 1 *If a is a Non-multiplexing Allocation in which the vector $a_i = (a_{i1}, a_{i2}, \dots, a_{im})$ is the unique initial optimal solution of task S_i for each $i \in [1..n]$, a_i is the optimal strategy of task S_i for each $i \in [1..n]$ and a is the unique equilibrium of the game.*

Proof Because of $\forall a_{ij} : (a_{ij} \leq 1) \wedge (\sum_{i=1}^{n} a_{ij} \leq 1)$, there is no multiplexed resource assignment. All strategies of the initial optimal solutions in the game are uncompetitive. So the *initial optimal solution* is the *final optimal solution* of each task. Precisely, there does not exist an allocation a'_i which satisfies $u_i(a'_i) \geq u_i(a_i)$ for all $i \in [1..n]$. □

5.2 Multiplexing

Lemma 1 *Let $a_i = (a_{i1}, \ldots, a_{ip}, \ldots, a_{iq}, \ldots, a_{im})$ be the initial optimal allocation obtained from independent optimization. $a'_i = (a_{i1}, \ldots, a'_{ip}, \ldots, a'_{iq}, \ldots, a_{im})$ denotes an evolutionary allocation according to a, where $a_{ip} \neq a'_{ip}$ and $a_{iq} \neq a'_{iq}$. Utility $u_i(a'_i) \leq u_i(a_i)$ for all $i \in [1..n]$.*

The proof is essentially described in Sect. 4.

Definition 2 (Multiplexing Allocation) Given that a resource allocation game has n tasks and m resources, each task consists of multiple same subtasks. If allocation matrix a satisfies the condition $\exists j : (a_{ij} \leq 1) \wedge (\sum_{i=1}^{m} a_{ij} > 1)$ for $i = [1..n], a$ is a *Multiplexing Allocation*.

From Definition 2, the occasion that at least one resource is assigned to two or more subtasks must exist in a multiplexing allocation. There are two possibilities: (1) Multiplexing Allocation does not influence the utilities of multiplexed tasks (case 1), and (2) Multiplexing Allocation decreases the utilities of Multiplexed tasks (case 2).

5.2.1 Case 1

Proposition 2 *Given a Multiplexing Allocation a in which the vector $a_i = (a_{i1}, a_{i2}, \ldots, a_{im})$ is the unique initial optimal solution of task S_i for each $i \in [1..n]$, to an arbitrary multiplexing assigned resource R_u, if there exist a $v \in [1..m]$ for a multiplexing allocation vector a_i, such as:*

(1) $a_{iv} = 1$,
(2) $\sum_{i=1}^{n} a_{iv} = 1$,
(3) $t_{iv} = \max_{j=[1..m]} t_{ij}$, *i.e.*, $t_{iu} < t_{iv}$,

a_i is the optimal strategy of task S_i for each $i \in [1..n]$ and a is the unique equilibrium of the game.

Proof From the utility function of tasks, it can be inferred that (1) the time cost of task S_i $\max_{t_{ij} \in t_i} \{t_{ij}\}$ because of $t_{iv} = \max_{j=[1..m]} t_{ij}$ and $u \neq v$, and (2) the expense $\sum_j e_{ij}$ has also not changed. Therefore, utility of task S_i is not influenced by the multiplexing. From Lemma 1, it is known that there is no change. The result is the same for a Non-multiplexing Allocation. □

5.2.2 Case 2

Multiplexing Allocation will decrease the utility of the multiplexed task in which one of the multiplexed subtasks has the longest completion time. It is possible for the task to minimize loss by reallocating the multiplexed subtask with the longest completion time. Definition 3 gives a precise definition to characterize the efficiency loss of a reallocation in which only one subtask is reassigned compared to its prior allocation. Our objective is to minimize the efficiency loss of each evolutionary step.

Table 1

Algorithm 1. SPELR minimization

```
Input: matrix $a$; task ID $i$; $p$;
Output: $q$
$MinSingle(a, i, p)$
{
  for all $j \in [1..m] \wedge j \neq p$
    compute $SPELR_{ralloc(i,p,j)}$;
  if $\min\{SPELR_{ralloc(i,p,j)}\} < 0$
    $q = \min_j\{SPELR_{ralloc(i,p,j)}\}$;
  else
    $q = -1$;
  return $q$
}
```

Definition 3 (Single Participant Efficiency Loss of a Reallocation, SPELR): If a task shifts a subtask from one resource to another, the SPELR is the amount by which its prior utility is greater than the later one.

For example, let a_i be the prior allocation of task S_i in which $t_{ip} = \max_{j\in[1..m]}\{t_{ij}\}$, a'_i be the new allocation after a subtask reassigned from R_p to R_q, where $a'_{ip} = 0$ and $a'_{iq} = a_{ip} + a_{iq}$. And let function $ralloc(i, p, q)$ denote this reallocation. Then, the SPELR of $ralloc(i, p, q)$ is $u_i(a_i) - u_i(a'_i)$.

So, the objective is to find such q which satisfies $\min_{q\in[1..m]\wedge q\neq p}\{ralloc(i, p, q)\}$ when performing a reallocation for task S_i. Table 1 shows the algorithm of finding a minimum SPELR.

In the competitive resource allocation game, task's S_i changing strategy will influence the utilities of other tasks. It is possible that more than one task assign one subtask to the same resource and the completion time of these subtasks is the maximum one of the same class of subtasks respectively. In this special case, in order to determine which task is reallocated first, the global efficiency losses of their minimum SPELR should be compared. The global efficiency loss of a reallocation is described in Definition 4.

Definition 4 (Global Efficiency Loss of a Reallocation, GELR): If a task shifts a subtask from one resource to another, the GELR is the amount that remains after the sum of prior utility of all tasks is subtracted from the later one.

For example, let $a = (a_1, \dots, a_i, \dots, a_m)^T$ be the allocation matrix before a reallocation, a_i be the allocation vector of task S_i which needs to shift a subtask, $a' = (a_1, \dots, a'_i, \dots, a_m)^T$ be the allocation matrix before a reallocation by which a_i changes to a'_i with others unchanged. Then, the GELR of this reallocation is $\sum_{a_i\in a} u_i(a_i) - \sum_{a'_i\in a'} u_i(a'_i)$.

Table 2

Algorithm 2. GELR minimization

```
Input: matrix $a$; Resource $j$;
Output: task ID $i$
$MinGlobal(a, j)$
{
  Get multiplexing task set $mts$ in resource $R_j$
  for all tasks $k \in mts$
    {
      $q = MinSingle(a, k, j)$;
      if $q \neq -1$
      {
        $ralloc(k, j, q)$; // suppose reallocate
        If $u_k(a_k) - u_k(a'_k) < 0$
          add $k$ to negative SPELR task set $nsts$;
      }
    };
    If $nsts$ is null
      $i = -1$;
    else
      $i = \min_k\{GELR_k\}$ for all $k \in nsts$;
    Return $i$
}
```

When there is more than one task with negative SPELR, for the objective of fairness, the task which incurs minimum GELR is chosen to perform reallocation. Table 2 shows the algorithm of finding a minimum GELR. When determining which task should perform reallocation, minimizing SPELR and GELR avoids starvation and keeps fairness.

Based on Algorithm 2, the evolutionary optimization algorithm (as described in Table 3) is designed to achieve the final optimal solution for the resource allocation game.

Note that the deadline of the completion time of each task is given. And the completion time is the sum of the execution time and the communication time, in which the execution time varies as task's resource allocation strategy changes. Assume the communication time is a fixed value that does not vary as task's resource allocation strategy changes. Therefore, it is reasonable to let T_i denote the deadline of execution time, and M_i denote the maximal expense that can be afforded for task S_i.

For all tasks, their deadlines of execution time and maximal expense are the constrained conditions of the evolutionary optimization.

Let a be the final optimal allocation of the game. Since tasks' budgets are not taken into consideration in the evolutionary optimization process, a may not be a feasible allocation. If there exists a task S_i for which $\max\{t_{ij}\} > T_i$ or $\sum_{j=1}^{m} e_{ij} > M_i$, the game has no fairness solution.

Table 3

Algorithm 3. Evolutionary optimization

```
Input: matrix $a$;
$EvolOptimize(a)$
{
  $i = 1$
  flag = True;
  while flag{
    if $i == 1$ flag = False;
    obtain multiplexing resource vector of task $S_i$, denote as $ms$;
    order $ms$ by $t_{ij}$ descent;
    for all $j \in ms${
      $q = MinGlobal(a, j)$;
      if $q \neq -1${
        $p = MinSingle(a, q, j)$;
        execute $ralloc(q, j, p)$;
        flag = True;
      }//end of if
    }//end of for
    if $(i == n)${
      if $flag == False$ exit;
      else $i = 1$;
    }//end of if
    else
      $i = i + 1$;
  }//end of while
}//end of $EvolOptimize$
```

Proposition 3 *If the final evolutionary solution satisfies the constrained conditions of all tasks, it forms a Nash equilibrium of the resource allocation game.*

Proof Let a_i be the strategy of an arbitrary task S_i, a_{-i} be the strategies of the remaining ones. Assume S_i has another feasible strategy a_i', $u_i(a_i', a_{-i}) > u_i(a_i, a_{-i})$. It is reasonable to consider a_i' is the evolution of a_i' with only one-step evolutionary optimization. In other words, there is only one subtask assignment which is different from a_i in a_i'. Without loss of generality, let $a_i = (a_{i1}, \ldots, a_{ip}, \ldots, a_{iq}, \ldots, a_{im}), a_i' = (a_{i1}, \ldots, a_{ip}', \ldots, a_{iq}', \ldots, a_{im})$. Suppose $a_{ip} = 1$ shifts to a_{iq}. There are two cases before the shifting.

(1) $$\sum_{i=1}^{n} a_{ip} = 1 \text{ and } \sum_{i=1}^{n} a_{iq} = 0.$$

In this case, resource R_p is solely assigned to task S_i since $\sum_{i=1}^{n} a_{ip} = 1$. Meanwhile, resource R_q is not assigned to any task since $\sum_{i=1}^{n} a_{iq} = 0$. Since the allocations of R_p and R_q are non-multiplexing, it can be inferred that task S_i increases its utility without interfering other tasks, i.e. $u_i(a_i)$ increases while $u_{-i}(a_{-i})$ is not changing, by shifting a subtask from R_p to R_q. Because $a_{ip} = 1$ is the entry of the final evolutionary allocation, according to the processes of initial and evolutionary optimizations, it is known that there does not exist an R_q which makes $u_i(a'_i) > u_i(a_i)$ and $u_{-i}(a'_{-i}) = u_{-i}(a_{-i})$ for all $\{q : q \in [1..m] \wedge (q \neq p) \wedge (\sum_{i=1}^{n} a_{iq} = 0)\}$, i.e. $u_i(a'_i, a_{-i}) > u_i(a_i, a_{-i})$. It is contrary to the above assumption.

(2) $$\sum_{i=1}^{n} a_{ip} > 1 \text{ or } \sum_{i=1}^{n} a_{iq} \geq 1.$$

In this case, suppose that shifting a subtask from R_p to R_q increases $u_i(a_i)$, i.e. $u_i(a'_i) > u_i(a_i)$. It will cause $u_{-i}(a_{-i})$ to decrease since $\sum_{i=1}^{n} a_{iq} \geq 1$. Let $\Delta u_i(a_i)$ denote the increment and $\Delta u_{-i}(a_{-i})$ denote the decrement. By the process of the above evolutionary optimization algorithm, $|\Delta u_i(a_i)| < |\Delta u_{-i}(a_{-i})|$ is known. Therefore, $u_i(a'_i, a_{-i}) < u_i(a_i, a_{-i})$. It is also contrary to the above assumption. $\square$

6 Experiment and performance evaluation

To demonstrate the effectiveness of the evolutionary optimization algorithm, the use case designed in Sect. 3 is also used along with the initial optimal results obtained in Sect. 4.4.

It is obvious that initial the result

$$a = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

is a multiplexing allocation, where resources R_3, R_4 and R_5 are multiplexed. According to this allocation, for each task, its maximum subtask execution time is produced by multiplexed subtask. In the execution time matrix

$$t_{ij} = \begin{pmatrix} 0 & 0 & 0 & 7 & 9 \\ 0 & 0 & 7.2 & 6 & 8.4 \\ 4 & 3.5 & 6.4 & 0 & 7.2 \end{pmatrix},$$

$$t_{i5} = \max_{j \in [1..5]}\{t_{ij}\} \text{ for each row in the } t_{ij}.$$

Based on this context, the evolutionary optimization algorithm (as described in Table 3) is run to seek a more optimal allocation (or set of strategies). The evolutionary steps are as below.

Step 1: Find valid SPELR and GELR. If not found, then exit.

From a and t_{ij}, $j = 5$ is chosen to reallocate. To strategy a_1, no negative SPELR can be found after computing minimum SPELR, i.e. current a_1 is the best strategy of task S_1. To strategy a_2, the valid minimum SPELR and GELR are obtained when objective $i = 2$, i.e. when task S_2 changes its strategy $a_2 = (0, 0, 1, 1, 1)$ to $a'_2 = (0, 1, 1, 1, 0)$, the SPELR of task S_2 is -0.001 and the GELR is -0.0062. To strategy a_3, no negative SPELR can be found.

Table 4 Utility comparison

Task	Initial strategy	Final strategy	Initial utility	Final utility
S_1	(0,0,0,1,1)	(0,0,0,1,1)	0.0469	0.0518
S_2	(0,0,1,1,1)	(0,1,1,1,0)	0.0403	0.0413
S_3	(1,1,1,0,1)	(1,1,1,0,1)	0.0400	0.0403
		Total utility	0.1272	0.1334

Step 2: Perform reallocation and compute new utilities for all tasks. According to step 1, the new allocation matrix is

$$a' = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

And the new utilities are $u'_1(a'_1) \simeq 0.518$, $u'_2(a'_2) \simeq 0.403$ and $u'_3(a'_3) \simeq 0.412$, respectively. It is obvious that the new allocation is approximated more to optimal than to initial allocation.

Step 3: Go to step 1. Repeat reallocating till no valid SPELR and GELR is found.

In this use case, the allocation matrix

$$a' = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

is the final result. The corresponding strategies of the three tasks are $a'_1 = (0, 0, 0, 1, 1)$, $a'_2 = (0, 1, 1, 1, 0)$ and $a'_3 = (1, 1, 1, 0, 1)$. Table 4 lists the comparison between the initial and evolutionary allocations.

The result shows that all participants (tasks) increase their utilities after evolutionary optimization. As a' is an equilibrium state of the resources allocation game (proved in Sect. 5.2), no task can choose a new strategy a'_i which increases its own utility but decreases other tasks' utilities. In this game, the total utility may be not optimal. But, the evolutionary result is most approximate to optimal results when the fairness is guaranteed.

The computation of the proposed method in this paper is mainly produced by the initial optimization and evolutionary optimization. The initial optimization is a *Binary Integer Programming* problem. The time complexity for completing the initial optimization of all tasks is $O(mn^2)$. The time complexity of the evolutionary optimization, as described in Sect. 5, is $O(mn \log m)$. The two steps' optimizations are serial. Therefore, the time complexity of the whole solution is $\max\{O(mn \log m), O(mn^2)\}$.

The resource allocation problem considered in paper has been studied by Nisan and Ronen in [26, 27], where it was shown that the approximation ratio of mechanisms is between 2 and n. Christodoulou et al. [11] improved the lower bound to

$1+\sqrt{2}$ for three or more resources. But they all studied this problem from machinery point of view. The approximation mechanism proposed in this paper attempts to solve practical problem from the user's point of view The real importance of our result lies in the fact that it is more practical with acceptable time complexity than mechanisms in [3, 7, 11, 12, 18, 20, 21, 26, 27, 29, 31]. The approximation ratio is mainly not concerned in this paper although it is very important for the solutions of NP-complete problems.

7 Conclusions

A game-theoretic method for scheduling cloud-based computing services with collaborative QoS requirements has been presented. In the computational service (or resource) market, a cost is incurred at each service that depends on the amount of computation. And each computing task has multiple dependent and homogeneous subtasks which are sensitive and interested in execution time. Game theory is used to find approximated solutions of this problem. Firstly, a *Binary Integer Programming* method is proposed to obtain the initial independent optimization, which does not take the multiplexing of resource assignments into consideration. Then, based on the initial result, an evolutionary mechanism is designed to achieve the final optimal and fair solution. By introducing concepts of SPELR and GELR, three algorithms for strategy evolution of all participants considering minimizing of their efficiency losses are designed. It is demonstrated that Nash equilibrium always exists if the resource allocation game has feasible solutions.

The optimization problem considered in this paper relates to a large proportion of cloud-based computing services. The method may be a useful analytical tool for shedding light on seeking optimal scheduling solution for the complex and dynamic problems that can be divided into multiple cooperative subtasks in many cloud-based computing and data store services.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China under Grants Nos. 60673179 and 60803161, Natural Science Foundation of Zhejiang Province of China under Grant No. Z106727.

References

1. Al-Ali R, Amin K, von Laszewski G, Rana O, Walker D, Hategan M, Zaluzec N (2004) Analysis and provision of QoS for distributed grid applications. J Grid Comput 2(2):163–182
2. Amoura AK, Bampis E, Kenyon C, Manoussakis Y (2002) Scheduling independent multiprocessor tasks. Algorithmica 32:247–261
3. An B, Douglis F, Ye F (2008) Heuristics for negotiation schedules in multi-plan optimization. In: Proc of the seventh international joint conference on autonomous agents and multi-agent systems, 2008, pp 551–558
4. An B, Vasilakos AV, Lesser V (2009) Evolutionary stable resource pricing strategies. In: ACM SIGCOMM 2009, Barcelona, Spain, August 17–21, 2009
5. Andelman N, Azar Y, Sorani M (2005) Truthful approximation mechanisms for scheduling selfish related machines. In: STOC 2005, pp 69–82
6. Borst S, Boxma O, Groote JF, Mauw S (2003) Task allocation in a multi-server system. J Sched 6:423–436

7. Boyera WF, Hura GS (2005) Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. J Parallel Distrib Comput 65:1035–1046
8. Briest P, Krysta P, Vöcking B (2205) Approximation techniques for utilitarian mechanism design. In: STOC 2005, pp 39–48
9. Buyya R, Abramson D, Giddy J, Stockinger H (2002) Economic models for resource management and scheduling in grid computing. Special issue on grid computing environments. J Concurr Comput: Pract Exp (CCPE) 14:13–15
10. Christodoulou G, Koutsoupias E, Kovacs A (2007) Mechanism design for fractional scheduling on unrelated machines. In: ICALP 2007
11. Christodoulou G, Koutsoupias E, Vidali A (2007) A lower bound for scheduling mechanisms. In: SODA 2007, pp 1163–1169
12. Collins DE, George AD (2001) Parallel and sequential job scheduling in heterogeneous clusters: a simulation study using software in the loop. Simulation 77:169–184
13. Dobzinski S, Nisan N, Schapira M (2005) Approximation algorithms for combinatorial auctions with complement-free bidders. In: STOC 2005, pp 610–618
14. Dogan A, Özgüner F (2002) Scheduling independent tasks with QoS requirements in grid computing with time-varying resource prices. In: CCGRID 2002, pp 58–69
15. Doulamis N, Doulamis A, Litke A, Panagakis A, Varvarigou T, Varvarigos E (2007) Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing. Comput Commun 30:499–515
16. Hochbaum DS (1996) Approximation algorithms for NP-hard problems. PWS Publishing, Boston
17. Iverson MA, Ozguner F, Potter L (1999) Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. IEEE Trans Comput 48(12):1374–1379
18. Karatza HD, Hilzer RC (2003) Parallel job scheduling in homogeneous distributed systems. Simulation 79(5–6):287–298
19. Koole G, Righter R (2008) Resource allocation in grid computing. J Sched 11:163–173
20. Korkhov VV, Krzhizhanovskaya VV, Sloot PMA (2008) A grid-based virtual reactor: parallel performance and adaptive load balancing. J Parallel Distrib Comput 68:596–608
21. Korkhova VV, Moscicki JT, Krzhizhanovskaya VV (2009) Dynamic workload balancing of parallel applications with user-level scheduling on the grid. Future Gener Comput Syst 25:28–34
22. Lavi R, Swamy C (2007) Truthful mechanism design for multi-dimensional scheduling via cycle-monotonicity. In: EC 2007
23. Lenstra JK, Shmoys DB, Tardos E (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46(1):259–271
24. Li K (2004) Experimental performance evaluation of job scheduling and processor allocation algorithms for grid computing on metacomputers. In: IPDPS 2004, pp 170–177
25. Mualem A, Schapira M (2007) Setting lower bounds on truthfulness. In: SODA 2007, pp 1143–1152
26. Nisan N, Ronen A (1999) Algorithmic mechanism design (extended abstract). In: STOC 1999, pp 129–140
27. Nisan N, Ronen A (2001) Algorithmic mechanism design. Games Econ Behav 35:166–196
28. Ranganathan K, Ripeanu M, Sarin A, Foster I (2004) Incentive mechanisms for large collaborative resource sharing. In: CCGrid 2004, pp 1-8
29. Schoneveld A, de Ronde JF, Sloot PMA (1997) On the complexity of task allocation. Complexity 3:52–60
30. Schopf JM (2003) Ten actions when grid scheduling. In: Nabrzyski J, Schopf JM, Weglarz J (eds) Grid resource management: state of the art and future trends. Kluwer, Dordrecht (Chap 2)
31. Sonmez OO, Gursoy A (2007) A novel economic-based scheduling heuristic for computational grids. Int J High Perform Comput Appl 21(1):21–29
32. Wolski R, Plank JS, Brevik J, Bryan T (2001) G-commerce: Market formulations controlling resource allocation on the computational grid. In: Proceeding of IPDPS 2001, p 46