Future Generation Computer Systems 37 (2014) 478–487

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Integrating QoS awareness with virtualization in cloud computing systems for delay-sensitive applications

Jenn-Wei Lin [a,*], Chien-Hung Chen [a], Chi-Yi Lin [b]

[a] Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, Tamkang University, Taiwan, ROC

HIGHLIGHTS

- We investigate the QoS-aware virtual machine placement (QAVMP) problem.
- We formulate the QAVMP problem as an Integer Linear Programming (ILP) model.
- We propose a polynomial-time heuristic algorithm to efficiently solve the QAVMP problem.
- A bipartite graph is used to model all possible placement relationships of virtual machines.
- The proposed heuristic algorithm can maximize the profit of cloud provider.

ARTICLE INFO

Article history:
Received 27 December 2012
Received in revised form
3 October 2013
Accepted 13 December 2013
Available online 9 January 2014

Keywords:
Cloud computing
Virtualization
Quality of service
Technique integration
Heuristic algorithm

ABSTRACT

Cloud computing provides scalable computing and storage resources over the Internet. These scalable resources can be dynamically organized as many virtual machines (VMs) to run user applications based on a pay-per-use basis. The required resources of a VM are sliced from a physical machine (PM) in the cloud computing system. A PM may hold one or more VMs. When a cloud provider would like to create a number of VMs, the main concerned issue is the VM placement problem, such that how to place these VMs at appropriate PMs to provision their required resources of VMs. However, if two or more VMs are placed at the same PM, there exists certain degree of interference between these VMs due to sharing non-sliceable resources, e.g. I/O resources. This phenomenon is called as the VM interference. The VM interference will affect the performance of applications running in VMs, especially the delay-sensitive applications. The delay-sensitive applications have quality of service (QoS) requirements in their data access delays. This paper investigates how to integrate QoS awareness with virtualization in cloud computing systems, such as the QoS-aware VM placement (QAVMP) problem. In addition to fully exploiting the resources of PMs, the QAVMP problem considers the QoS requirements of user applications and the VM interference reduction. Therefore, in the QAVMP problem, there are following three factors: resource utilization, application QoS, and VM interference. We first formulate the QAVMP problem as an Integer Linear Programming (ILP) model by integrating the three factors as the profit of cloud provider. Due to the computation complexity of the ILP model, we propose a polynomial-time heuristic algorithm to efficiently solve the QAVMP problem. In the heuristic algorithm, a bipartite graph is modeled to represent all the possible placement relationships between VMs and PMs. Then, the VMs are gradually placed at their preferable PMs to maximize the profit of cloud provider as much as possible. Finally, simulation experiments are performed to demonstrate the effectiveness of the proposed heuristic algorithm by comparing with other VM placement algorithms.

1. Introduction

Cloud computing provides scalable computing and storage resources via the Internet. User can use these infrastructure resources (computing and storage resources) based on a pay-per-use basis. This delivery model of *infrastructure as a service (IaaS)* has been provided by several cloud providers, such as Amazon Elastic Compute Cloud (EC2) [1], Google Compute Engine (GCE) [2], and GoGrid [3], etc. In the IaaS delivery model, the key technology is how to efficiently virtualize the computing and storage resources of physical machines (PMs) to provision a large number of virtual machines (VMs). Each customer (user) can rent a VM from the cloud provider to execute his/her application. Amazon EC2

* Corresponding author. Tel.: +886 229053855.
E-mail address: jwlin@csie.fju.edu.tw (J.-W. Lin).

0167-739X/$ – see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.12.034

provides different instance types of VMs to meet the computing needs of users [4]. Several virtualization technologies (e.g. Xen [5], VMware [6], KVM [7]) have been used in cloud computing systems. Using the virtualization technologies, more than one VM can be created in the same PM, each of which acquires its required resources by slicing a portion of resources from the PM. For example, in Amazon EC2, assume that a small instance VM i and a large instance VM j are created on the same PM p. The PM p will be virtualized to form a machine with 1 EC2 Compute Unit, 1.7 GB memory, 160 GB instance storage for VM i and a machine with 4 EC2 Compute Units, 7.5 GB memory, 850 GB instance storage for VM j. However, the existing virtualization technologies cannot slice all hardware resources of a PM. Some types of hardware resources are non-sliceable, which are called the non-sliceable resources. For example, in [8], it clearly indicated that the disk I/O, network I/O, and L2 cache are the non-sliceable resources. In [9], the authors also claimed that I/O virtualization is a big challenge, and there is no ideal solution. It is common that two or more VMs on the same PM will contend the non-sliceable resources. In a VM, the application cannot be executed in a fully isolated computing environment. It is inevitable that if VMs i and j are created in the same PM, the application running on VM i will affect the performance of the application running on VM j. It means that, for the VMs on the same PM, there exists performance interference among the VMs. We use the term *VM interference* to describe this phenomenon. For two VMs on different PMs, their performance may be also interfered with each other if their corresponding applications are dependent with each other. This type of VM interference is not discussed in this paper since the issue is not introduced due to the contention of non-sliceable resources of a PM.

For a network I/O (delay-sensitive) application, its QoS requirement is usually defined how much time is taken to process a network I/O request. If the process time of the network I/O request is equal to or less than a pre-specified time requirement in the service level agreement (SLA). The QoS requirement of the application is satisfied. From the view point of a VM, the QoS requirement of its running application can be preliminary met by allocating appropriate resource to the VM. However, if multiple VMs are created on the same PM, each VM may incur certain degree of performance degradation due to the VM interference. In such situation, the VM is difficult to guarantee that its computing environments can continuously meet the QoS requirement of its running application. Even if there are few VMs on the same PM, it is also an inappropriate placement to allocate two or more VMs with high-QoS applications at the same PM. From the cloud provider perspective, if the created VMs cannot provide the computing environments to satisfy the QoS requirements of running applications, it will pay penalties due to violating the service level agreements (SLAs) with users.

In this paper, we investigate how to integrate QoS awareness with virtualization for efficiently performing delay-sensitive applications in cloud computing systems. Specifically, we called it the *QoS-aware virtual machine placement (QAVMP)* problem. In cloud computing, most of applications are with the data-intensive feature to frequently read and write data. The data is also stored based on the distributed manner. Therefore, a data access will involve one or more network I/O operations which is processed by the non-sliceable resource: network interface card. With involving the non-sliceable resource, the data access of one applications will be affected by other applications due to the VM interference. For a delay-sensitive application, if its data access delay is larger than its expected data response time stated in the service level agreement (SLA), the QoS requirement of this application will be violated. Compared to previous VM placement strategies [10–19], our QAVMP problem considers the QoS requirements of applications and VM interference in addition to the resource utilization of PMs. With the three concerned factors in the QAVMP problem, its optimal solution is difficult to be found. By integrating the three concerned factors into the profit metric of a cloud provider, we can formulate the QAVMP problem as an integer linear programming (ILP) model to find its optimal solution. However, long computation time will be required to find the optimal solution. To seek a time-efficient solution to the QAVMP problem, we also propose a heuristic algorithm with polynomial time to solve the problem. In the proposed heuristic algorithm, a bipartite graph is first used to model all the possible placement relationships between VMs and PMs. Based on the bipartite graph, each VM is gradually placed at its preferable PM to maximize the profit metric of the cloud provider as much as possible. Overall, the main contributions of this paper are summarized as follows.

- Unlike prior VM placement strategies in [10–19], our QAVMP problem additionally takes the VM interference effects and Qos requirements of applications into the VM placement. The proposed algorithm can reduce the VM interference and avoid violating the QoS requirements of applications after the VM placement.
- The proposed QAVMP algorithm also considers the dynamical VM creation requests. The VM creation requests arrive dynamically without any knowledge of future requests. Before creating a number of new VMs on a PM, the PM may possibly hold several existing VMs. The VM interference also exists among the existing VMs and the new VMs.
- We use an ILP model to formulate the optimal solution of the QAVMP problem.
- We propose a heuristic placement algorithm to efficiently solve the QAVMP problem in polynomial time.

The rest of the paper is organized as follows. In Section 2, we introduce the preliminaries of this paper. In Section 3, we propose an ILP model to solve the QAVMP problem optimally. In addition, the heuristic algorithm of the QAVMP problem is also presented. In Section 4, we conduct simulation experiments to evaluate the performance of the proposed heuristic algorithm. Finally, Section 5 concludes the paper.

2. Preliminaries

In this section, we give brief introduction to virtualization techniques and describe the system model used in our paper. Furthermore, we also review the previous VM placement schemes.

2.1. Virtualization techniques

Virtualization techniques can make hardware resources of a physical machine (PM) to be shared with multiple operating systems. It respectively provides a virtual environment for each operating system in the PM. Therefore, an operating system in the virtual environment can be viewed as an independent virtual machine (VM). There are two types of virtualization techniques: full virtualization and para-virtualization. The full virtualization emulates a complete hardware environment to be compatible with any operating systems. This approach is very popular, because it does not have to modify the original operating systems. However, it increases some overhead due to the virtualization of the hardware devices. Para-virtualization is another approach that can improve the efficiency of hardware virtualization. Using para-virtualization, the operating systems are aware of working in a virtual environment. In other words, the operating systems must be modified for virtualization. Regardless of adopting which one of the virtualization techniques, it requires a monitor to handle resource allocation for the VMs.

2.2. Xen hypervisor

Xen hypervisor is an open source virtual machine monitor [20,21]. As shown in Fig. 1, Xen architecture consists of Xen hypervisor and guest domains. The Xen hypervisor can protect guest

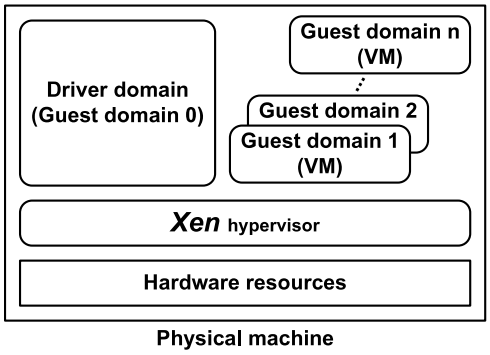

Fig. 1. Xen I/O architecture.

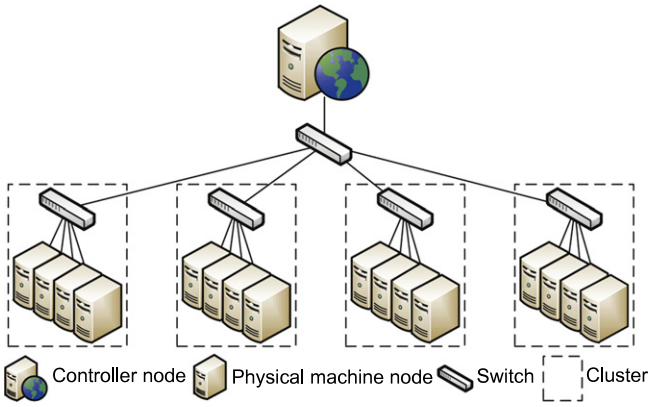

Fig. 2. System model.

domains to be isolated from each other. It provides an abstraction layer between guest domains and actual hardware of the PM, which performs functions such as CPU scheduling and memory allocations for the guest domains. The Xen hypervisor can slice the resources of a PM to offer the guest domains (VMs) in a PM. In Xen, the guest domain can be imaged as a VM. The guest domains cannot directly access the I/O devices of the PM, e.g., disk and network interface card. A special guest domain called the driver domain has the I/O drivers of the PM. The driver domain performs I/O operations on behalf of guest domains. All I/O traffic of guest domains is processed through the driver domain. All I/O requests of guest domains have to wait until that the driver domain is running. For example, when a guest domain issues a network I/O request for a network packet, the request will be sent to the driver domain as an event. On receiving the event, the driver domain forwards this packet to the network interface card.

Xen Cloud Platform is a server virtualization platform, which extends the Xen architecture to adapt the systems with a large number of PMs. Using this virtualization platform, one of the PMs is selected to collect hardware resource information across the whole system and to deploy multiple VMs into the PMs of the system.

2.3. System model

We refer to the two-level cloud architecture to investigate the QAVMP problem. The cloud system consists of a single controller node and a number of physical machine (PM) nodes, as shown in Fig. 2. The referred system architecture is similar to the architecture of Xen Cloud Platform [22].

The PM nodes are distributed within a set of clusters. In each cluster, there is a switch to provide the communication between any two PM nodes in the same cluster and the communication with the controller node. The controller node mainly manages the resources of PM nodes and performs the VM placement. Each PM node periodically sends the amount of its available resources to the controller node. Using the available resource information, the VM placement algorithm is performed in the controller node.

2.4. Related work

For the VM placement issue, it has been discussed extensively in the literature [10–19], among them, the VM placement problem is usually transformed to the multiple knapsack problem. With the problem transformation, the ILP model corresponding to the VM placement can be easily formulated. Based on the derived ILP model, the optimal solution of the VM placement problem can be obtained.

In addition to considering the resource utilizations of PMs, the VM placement algorithm of [19] specially concerns the data transmission delay problem. The data transmission delay is defined as the data access time between storage and a VM. In cloud computing systems, the performance of data-intensive applications strongly depends on data transmission delay. It is required to take the data transfer delay into the VM placement. The work of [17,18] provided deep insight into the resource provisioning of VMs in cloud computing systems. In [17], an optimal virtual machine placement (OVMP) algorithm was proposed to provision the resources of VMs based on two provisioning plans: reservation and on-demand. In the reservation plan, the customers need to reserve the resources in advance. In the on-demand plan, the resources are dynamically provisioned to the customers based on the pay-per-use basis. The OVMP algorithm can optimally adjust the tradeoff between the advance reservation of resources and the dynamical allocation of on-demand resources. Based on the OVMP algorithm, the total cost of resource provisioning of VMs can be minimized in a multiple cloud provider environment. In addition, the algorithm also takes the demand and price uncertainties into the resource provisioning. Finally, in the OVMP algorithm, the optimal resource provisioning solution is obtained by formulating and solving a stochastic integer programming problem. To further improve the OVMP algorithm, the same authors proposed another optimal cloud resource provisioning algorithm in [18], called the OCRP algorithm. The OCRP algorithm extends the OVMP algorithm to provision resources of VMs in multiple provisioning stages. To solve the optimal resource provisioning in an efficient way, two different approaches (Benders decomposition and sample-average approximation) are applied in the OCRP algorithm instead of the stochastic programming approach. However, in [17,18], the VM placement was performed within a multi cloud provider environment, so only the coarse-grained VM placement information was given. Specifically, for each VM, the placement information only indicates which cloud provider hosts the VM, not the information about which PM hosts the VM. In addition, the VM interference is not discussed in [17,18].

Maximum:

$$\left(\sum_{i\in E}\sum_{j\in P} X_{ij}^{(e)} \times RP_i + \sum_{i\in N}\sum_{j\in P} x_{ij}^{(n)} \times RP_i\right) - \left(\sum_{i\in E}\sum_{j\in P} y_{ij}^{(e)} \times PP_i + \sum_{i\in N}\sum_{j\in P} y_{ij}^{(n)} \times PP_i\right), \tag{1}$$

subject to:

$$\forall i \in N, \quad \sum_{j\in P} x_{ij}^{(n)} \le 1, \tag{2}$$

$$\forall j \in P, \quad \sum_{i\in N} x_{ij}^{(n)} \times D_i^{(c)} \le R_j^{(c)}, \tag{3}$$

$$\forall j \in P, \quad \sum_{i\in N} x_{ij}^{(n)} \times D_i^{(m)} \le R_j^{(m)}, \tag{4}$$

$$\forall j \in P, \quad \sum_{i \in N} x_{ij}^{(n)} \times D_i^{(s)} \leq R_j^{(s)}, \tag{5}$$

$$\forall i \in E, \forall j \in P, IO_j > QoS_i, \quad X_{ij}^{(e)} \leq y_{ij}^{(e)}, \tag{6}$$

$$\forall i \in N, \forall j \in P, IO_j > QoS_i, \quad x_{ij}^{(n)} \leq y_{ij}^{(n)}, \tag{7}$$

$$\forall i \in E, j \in P, \quad X_{ij}^{(e)}, y_{ij}^{(e)} \in \{0, 1\}, \tag{8}$$

$$\forall i \in N, j \in P, \quad x_{ij}^{(n)}, y_{ij}^{(n)} \in \{0, 1\}. \tag{9}$$

The VM interference was investigated in [23–26]. The amount of VM interference cost depends on various factors, such as the types of applications running in VMs, the number of VMs placed at the same PM, the choice of the VM scheduling algorithm. In [27], the authors explored the relationship between VM (domain) scheduling in a virtual machine monitor (VMM) and I/O performance. Traditionally, different VMM schedulers only focused on how to fairly share the processor resources among domains. However, this can result in poor or unpredictable application performance in the I/O aspect. Using the VM technology, the authors combined different applications with processor-intensive, bandwidth-intensive, and latency sensitive types to be run concurrently in a PM, and then they evaluated the impacts of different schedulers on the processor and I/O performance. The study revealed that VMM schedulers do not achieve the same level of fairness for I/O-intensive applications as they run concurrently with compute-intensive applications. In [24,25], they investigated the impacts of different disk I/O schedulers for VMs. In [26], the experimental research focused on the measurements of network I/O performance interference among VMs. For the work of [23], it is based on the monetary cost to analyze the VM interference. In addition, the authors also proposed a pricing scheme to charge users according to their resource consumption excluding the effect of VM interference.

All the above VM interference literature only conducted extensive experiments to measure the interference cost in terms of the performance interference in various metrics (e.g., I/O execution and throughput, CPU utilization, disk latency, etc.). The literature did not discuss how to take the VM interference into the design of the VM placement algorithm.

3. QoS-aware VM placement

Theorem 1. *The QAVMP problem is an NP-hard problem.*

Proof. In the previous studies on the virtual machine placement (VMP) problem [10–19], the VMP problem is usually transformed to the multiple knapsack (MK) problem to maximize the resource utilizations of PMs in the creation of VMs. The MK problem is to solve the following problem. Given a set of knapsacks and a set of items, each knapsack has a capacity limit, and each item has a weight and a profit. The objective of the MK problem is to place items into the knapsacks such that the total profit of placed items is maximized and the total weight of items placed in each knapsack does not exceed the capacity limit of the knapsack.

For the VMP problem, it aims to place VMs into the PMs such that the profit of cloud provider is maximized. Each PM has limited resources to host a number of VMs, and each VM has its resource demand and the rental price.

In the VMP problem, each PM and VM can be regarded as a knapsack and an item in the MK problem, respectively. A PM has limited resources to host a number of VMs. A VM has its resource demand and the rental price. The objective of the VMP problem is to put VMs into the PMs as many as possible while not exceeding the resource limit of each PM. In addition, the cloud provider can obtain the maximum profit by placing VMs in PMs. With the above mappings, the VMP problem can be transferred to the MK problem.

However, the MK problem is well-known to be NP-hard [28]. The solution of the VMP problem will take much computational time. In addition to the resource utilization of PMs, our QAVMP problem additionally take the QoS requirements of applications and VM interference into the VM placement. Due to considering the above three factors, the QAVMP problem is more complicated than the VMP problem investigated in the previous literature. Therefore, the QAVMP problem is also NP-hard. □

Without considering the VM interference in the VM placement, the QoS requirements of applications running in VMs may be violated. Different from the traditional VM placement problem, our investigated QAVMP problem considers the following three factors in the VM placement: (1) the resource demand of a VM (2) the QoS requirements of applications, and (3) the VM interference (the interference among VMs). To integrate these three factors of the QAVMP problem into the VM placement, we are based on the profit perspective of a cloud provider to incorporate the above three concerned factors. Then, the QAVMP problem can be formulated as an ILP model to obtain the optimal solution. However, solving ILP will take considerable computational time. In this section, we will propose a heuristic algorithm with polynomial time to solve the QAVMP problem. The time complexity of the heuristic algorithm is also analyzed.

3.1. ILP model

The ILP is a known mathematical method for solving the optimal problems with following characteristics: a linear objective function, a number of linear constraints, and an integer solution set. Before formulating the QAVMP problem using the ILP model, we first make the following assumptions.

1. The cloud provider would like to create a number of new VMs in PMs concurrently. There are different resource demands for the new VMs. In addition, the new VMs have different rental prices for customers (users). Basically, the VM with a high resource demand has a high rental price.
2. If the provided computing environment (VM) cannot meet the QoS requirement of the user application, the cloud provider will return an amount of money to the user. Here, the penalty payment due to the QoS violation is stated in the SLA between the cloud provider and the user.
3. Before placing the new VMs, each PM already has held a certain number of existing VMs.

Based on the above assumptions, the ILP model for an instance P of the QAVMP problem can be expressed as Eqs. (1)–(9). In the ILP model, the objective function is to maximize the profit of the cloud provider after placing the new VMs in PMs. Table 1 lists the notations used to express the ILP model. Basically, the cloud provider would like to make PMs hold VMs as many as possible to generate more revenue. As increasing the number of VMs in a PM, there is a higher level of adverse effects on the performance of applications running on the VMs. This will possibly increase the penalty payment of the cloud provider in the VM provisioning. There is a tradeoff between the revenue and the penalty payment in the VM provisioning.

In Eq. (1), the above tradeoff is quantified using the profit metric. The calculation of the profit metric includes two terms. The first main term represents the total revenue due to establishing the existing and new VMs. The second main term is the total penalty payment due to violating the QoS requirements of the applications running in VMs. The decision variable $X_{ij}^{(e)}$ keeps the placement statuses of existing VMs. The value of each $X_{ij}^{(e)}$ is known in advance. If $X_{ij}^{(e)}$ is 1, it represents that the existing VM i is placed at PM j; otherwise not. The second term is the generated revenue after placing a number of new VMs. The decision variable $x_{ij}^{(n)}$ stores the

Table 1
Notations.

Notation	Description
P	A set of physical machines in a cloud computing system.
E	A set of existing virtual machines that were created in the cloud computing system.
N	A set of new virtual machines to be created.
RP_i	The rental price of the virtual machine i.
PP_i	The penalty due to violating the QoS of the application in the virtual machine i.
$X_{ij}^{(e)}$	The {0, 1} variable indicates whether the existing virtual machine i was created in the physical machine j.
$x_{ij}^{(n)}$	The {0, 1} variable indicates whether the new virtual machine i is placed in the physical machine j.
$y_{ij}^{(e)}$	The {0, 1} variable indicates whether the QoS violation occurs when $X_{ij}^{(e)} = 1$.
$y_{ij}^{(n)}$	The {0, 1} variable indicates whether the QoS violation occurs when $x_{ij}^{(n)} = 1$.
$R_j^{(c)}$	The available compute units of the physical machine j.
$R_j^{(m)}$	The available memory space of the physical machine j.
$R_j^{(s)}$	The available storage space of the physical machine j.
$D_i^{(c)}$	The required compute units of the new virtual machine i.
$D_i^{(m)}$	The required memory space of the new virtual machine i.
$D_i^{(s)}$	The required storage space of the new virtual machine i.
IO_j	The average service time of a network I/O request in the physical machine j.
QoS_i	The QoS of the application running in the virtual machine i.
λ_j	The arrival rate of network I/O requests to the PM j.
μ_j	The mean service time of network I/O requests in the PM j.
σ_j^2	The variance of the service time distribution for network I/O requests in the PM j.
Se_vm	The selected VMs of PMs.
Sr_vm	The corresponding bipartite vertices of selected VMs.
Sr_link	The corresponding bipartite edges of selected VMs.

placement statuses of new VMs. Note that the value of each $x_{ij}^{(n)}$ is determined by the ILP formulation. If $x_{ij}^{(n)}$ is 1, it represents that the new VM i is placed at PM j; otherwise not. Each existing or new VM i has a different rental price RP_i. In Eq. (1), the sum of the first and second terms represents the total revenue by establishing the existing and new VMs. The third and fourth terms are the penalty payments due to violating the QoS requirements of the applications running in the existing and new VMs, respectively. As mentioned in Section 1, the QoS violation is determined by verifying the response time of a data access. For an existing (new) VM i in the PM j, if its data response time cannot meet the requirement of its running application, this event will be kept in $y_{ij}^{(e)}$ ($y_{ij}^{(n)}$). Note that each VM i also has a different penalty payment C_i. The value of each $y_{ij}^{(e)}$ ($y_{ij}^{(n)}$) is also determined by the ILP formulation. The sum of the third and fourth terms represents the total penalty payment.

There are a number of constraints for placing the new VMs in PMs. In Eq. (2), it limits that each VM can be only placed at a certain PM. In Eqs. (3)–(5), it denotes that each PM cannot hold too many VMs to exceed the amount of its available resources. Next, Eqs. (6) and (7) are used to determine whether the application running in a VM may incur the QoS violation. Note that the QoS metric in this paper is the data response time. The M/G/1 queuing model [29] is used to estimate the mean completion time of a data access issued from a VM. Then, the estimated completion time is compared with the expected data response time stated in the QoS requirement. Based on the comparison result, we can conclude whether the QoS violation is made or not. The details are described in the following subsection. Finally, in Eqs. (8) and (9), they set the value domains of the decision variables: $X_{ij}^{(e)}$, $x_{ij}^{(n)}$, $y_{ij}^{(e)}$, and $y_{ij}^{(n)}$ to be {0, 1}.

3.2. The QoS-violation assessment

The assessment of QoS violation has been stated in Section 1 by verifying the completion time of a data access issued from a VM. From the introduction of Xen in Section 2.2, we also know that each VM in a PM uses the driver domain of the PM to process all its data accesses. If two VMs i and j are allocated on the same PM p, their data accesses are processed by the driver domain of PM p in a certain sequence. The following assumptions are given for modeling the execution behaviors of data accesses, which are also made in [30–32].

1. We assume that the data accesses issued from each VM i follow an independent Poisson distribution associated with an average rate of λ_i (data access issued per unit of time unit).
2. The data accesses to the driver domain are processed based on the first-come-first-served (FCFS) basis.
3. In the driver domain, the service time of a data access follows an arbitrary distribution. A general distribution is used to model the service time of a data access. In the given general distribution, the average and variance of the service time are assumed to be μ and σ^2, respectively.

Based on the above assumptions, the M/G/1 queuing model can be as the analytical performance model for evaluating the performance of data accesses (workloads) in the driver domain. Based on the M/G/1 queuing model, we can use the Pollaczk–Khintchine (P–K) formula to estimate the mean completion time of a data access in the driver domain. Note that if there were k data accesses in the driver domain before a new data access, the completion time of this new data access includes the waiting time for completing these k existing data access and the service time of the new data access. In the given P–K formula [29], there are three key factors to estimate the completion time of a data access.

- The arrival rate of data accesses to the driver domain (λ).
- The mean service time of a data access in the driver domain (μ).
- The variance of the service time distribution for data access (σ^2).

As mentioned in the above assumptions, each VM is based on an independent Poisson process to data accesses. Therefore, the arrivals of data accesses to the driver domain follow a compound (pooled) Poisson process, in which the arrival rate parameter is equal to the sum of the individual rate parameters $\lambda_1+\lambda_2+\cdots+\lambda_n$ for all the VMs on the PM p. This property has been proved in the probability literature [29]. The VM can issue data accesses when it obtains the CPU time. The CPU allocation time for a VM needs to be taken into the above arrival rate calculation. In Eq. (10), it gives the arrival rate of data accesses to the driver domain of PM p if there are $|E|$ existing VMs on PM p and $|V|$ new VMs would like to be placed at PM p.

$$\lambda_j = \sum_{i \in E} \lambda_i^{(e)} \cdot r_i^{(e)} + \sum_{i \in N} \lambda_i^{(n)} \cdot r_i^{(n)}, \tag{10}$$

where $r_i^{(e)}$ and $r_i^{(n)}$ are the ratio of CPU time allocated to the ith existing (new) VM during a CPU unit time.

By substituting the three key factors into the PK-formula in Eq. (11), the completion time of a data access can be estimated. Finally, the estimated completion time of a data access is compared with the required data response time. If the former is larger, the QoS is violated (see Eqs. (6)–(7)); otherwise not.

$$IO_j = \frac{2\mu_j - \lambda_j + \lambda_j \mu_j^2 \sigma_j^2}{2\mu_j^2 - 2\lambda_j \mu_j}. \tag{11}$$

3.3. Heuristic algorithm

In the above ILP model, the solutions are obtained and put in the decision variables $x_{ij}^{(n)}$, $y_{ij}^{(e)}$ and $y_{ij}^{(n)}$. There are $|V| \times |P|$ decision variables for $x_{ij}^{(n)}$ and $y_{ij}^{(n)}$, as well as $|E| \times |P|$ decision variables for $y_{ij}^{(e)}$. Solving ILP is also known to be an NP-complete problem [28].

Resource and placement parameters of VMs

	VMs	VM1	VM2	VM3	VM4	VM5
Resource parameters	CPU compute unit (GHz)	1	2	2	4	4
	Memory space (GB)	2	4	8	4	4
	Storage space (GB)	250	250	500	250	250
Placement parameters	Rental prices	70	80	85	90	90
	QoS (the expected data response time)	≤3	≤2.3	≤2	≤1	≤0.8
	Network I/O request rate	4	5	6	6	7

Resource and placement parameters of PMs

	PMs	PM1	PM2	PM3
Resource parameters	CPU compute unit (GHz)	2	7	10
	Memory space (GB)	4	10	16
	Storage space (GB)	400	300	200
Placement parameters	μ (mean service time for processing a network I/O request)	10	9	8

Profits relative to placement relations between VMs and PMs

	VM1	VM2	VM3	VM4	VM5
PM1	65	25	N/A	N/A	N/A
PM2	64	60	49	62	34
PM3	66	77	83	87	86

Fig. 3. An example to demonstrate an QAVMP problem.

If we use ILP to obtain the optimal solution to the QAVMP problem, much computational time will be taken. Therefore, we propose a heuristic algorithm to efficiently solve the QAVMP problem. Before getting into the technical details of the heuristic algorithm, Fig. 3 gives a VM placement scenario to elaborate how the heuristic algorithm works. In Fig. 3, we give a VM placement scenario for elaborating how the operations of the QAVMP algorithm works. It assumes that there are three PMs in a cloud computing system, each of which has already held a number of existing VMs. The cloud provider would like to place five new VMs in these three PMs. In addition to giving the resource demands of VMs and resource capacities of PMs, we also give some additional parameters for VMs and PMs. These additional parameters are used for demonstrating how the QAVMP algorithm considers the QoS requirements of applications and the VM interference in the VM placement. The issue rate of data accesses and the mean service rate of data accesses are given for calculating the service time of a data access. To compare with the expected data response time, we can verify whether the QoS requirement of a VM is violated or not. If so, the corresponding penalty payment is subtracted from the rental price of the VM.

The basic operations of the heuristic algorithm are shown in Fig. 4, which consists of four phases: construction phase, contention phase, placement phase, and reformation phase. The latter three phases are repeated with a number of rounds. Based on Fig. 3, we demonstrate the detailed operations of the heuristic algorithm in the above four phases.

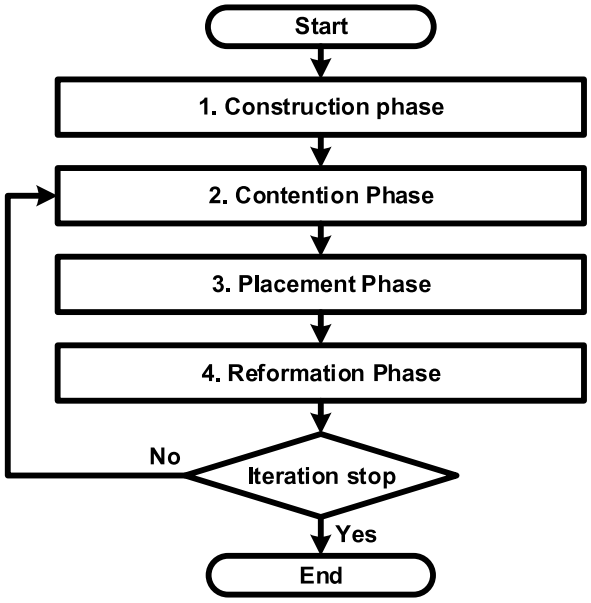

Fig. 4. A flowchart for the heuristic algorithm.

3.3.1. Construction phase

Initially, the heuristic algorithm establishes a bipartite graph to model all the possible placement relationships between new VMs and PMs. For the new VM i, if it has a link with the PM j, it represents that the PM j has enough resources to hold the VM i. Based the resource demands of new VMs and the resource capacities of PMs, a bipartite graph is established between the new VMs and the PMs. Then, the heuristic algorithm runs round by round to perform the three phases: contention, placement, and reformation. Fig. 5(a) first shows the bipartite graph formation for modeling the placement relationships between the VMs and PMs in Fig. 3.

3.3.2. Contention phase

In the contention phase, each PM j calculates the profit with respect to each of its possible new VM members. For the PM j, the VM i is one of its possible VM members if the VM i has a link with the PM j on the bipartite graph. Assume that a VM i is finally placed at the PM j. The correspondingly generated profit is calculated as follows.

$$profit = revenue - penalty, \tag{12}$$

where *revenue* is the rental price (revenue) of VM, and *penalty* is the introduced total penalty payment due to placing the VM at the PM. Here, the above mentioned P–K formula (see Eq. (11)) is used to verify the QoS violation of each VM in the PM. If so, the corresponding penalty payment of a VM is accumulated into *penalty*. After calculating the profits of all possible VMs, the PM selects a VM with the largest profit as the expected VM (the VM to be placed at the PM). For conveniently demonstrating the contention phase in Fig. 3, we use a table to directly show the profit calculation results of each PM in the first round. As shown in Fig. 5(a), the PM1 has 2 possible VM members: VM1 and VM2. In Fig. 5(b), the VM1 is selected as the expected VM of the PM1 since it has the largest profit 65. However, it is possible that two or more PMs select the same VM as their commonly expected VM. There is the expected VM selection conflict among PMs. To resolve the conflict, a collection set *Se_vm* is used. Whenever each PM selects

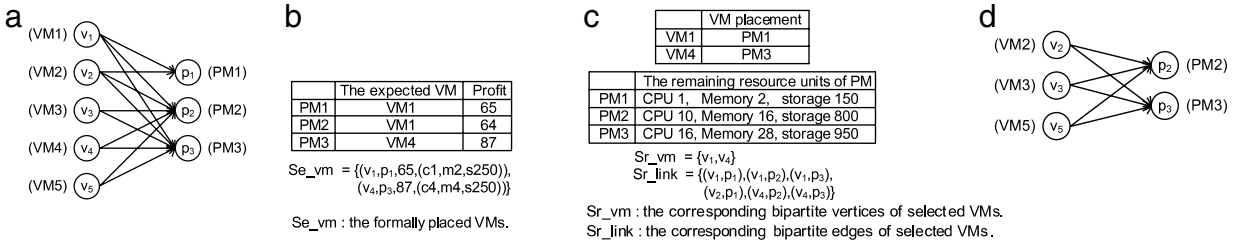

Fig. 5. Operations of the heuristic algorithm in the first round. (a) Initial bipartite graph. (b) Contention phase. (c) Placement phase. (d) Reformation phase.

an expected VM: vm, it puts a 4-tuple (vm, p_j, *profit*, $vm.resource$) in Se_vm. Assume that PMs p_1 and p_2 have the same expected VM v_i. If PM p_1 is priori to PM p_2 to put the corresponding 4-tuple of its expected VM in Se_vm, the PM p_2 will detect the conflict on the expected VM selection. In such case, the profit of the VM v_i in the PMs p_1 is compared with that in the PM p_2. If the later profit is larger, it represents that the placement of the VM v_i in the PM p_2 is superior to that in the PM p_1. Then, the 4-tuple (v_i, p_1, $profit_{(v_i,p_1)}$, $v_i.resource$) in Se_vm will be replaced by the 4-tuple (v_i, p_2, *profit*, $v_i.resource$). With the above operations, Se_vm can collect the placement information about some VMs. In Fig. 5(b), there is a conflict between PM1 and PM2 due to selecting the same expected VM. The VM1 is more suitable to be placed at the PM1 since the placement at PM1 can generate a larger profit (65) than that at PM2 (25). In the end of the contention phase, Se_vm provides the VM placement information. For example, Se_vm of Fig. 5(b) represents that VM1 and VM4 should be placed at PM1 and PM3, respectively.

3.3.3. Placement phase

In the placement phase, the set Se_vm is used to formally place VMs in their preferable PMs. Each VM in Se_vm is based on the objective of maximizing the profit to determine the PM where the VM should be placed. The profit maximization implies that the VM interference effects can be minimized. Note that the heuristic algorithm only places a portion of VMs in each round. After the placement phase, the placed VMs are required to be removed from the original bipartite graph. In Fig. 5(c), the set Sr_vm keeps the VMs to be removed. Due to placing some VMs, the resource capacities of some PMs are required to be updated. It is possible that the remaining resource capacity of a PM cannot continuously satisfy the resource demands of certain VMs. In such situation, the unsatisfied placement relationships are also required to be removed from the original bipartite graph. In Fig. 5(c), the set Sr_link keeps the un-satisfied placement relationships, such that which links are required to be removed from the original bipartite graph.

3.3.4. Reformation phase

In the reformation phase, the original bipartite graph is modified using the sets Sr_vm and Sr_link. Based on the new bipartite graph, the heuristic algorithm initiates a new round to continuously perform the placement of un-placed VMs. From Fig. 5(a), we can know that VM2, VM3 and VM5 are the un-placed VMs to be placed at the second round. The VM1 and VM4 were placed at the first round. These VM nodes are removed from the original bipartite graph. In addition, the original link between PM1 and VM2 in Fig. 5(a) is removed since the remaining resource capacity of PM1 cannot meet the resource demand of VM2.

Finally, the heuristic algorithm terminates under one of the following two conditions: (1) All VMs have been placed. (2) None PM can provide enough resources to hold anyone of un-placed VMs. Either of the above two conditions will cause that the new bipartite graph cannot be established between the up-placed VMs and PMs. In such case, the heuristic algorithm stops. More detailed operations are provided in Fig. 6.

Input: A set V of new VMs to be placed at a set P of PMs.
Output: QoS-aware VM placement for the new VMs.

1: **/* Construction phase */**
2: Use V and P to form a bipartite graph.
3: **while** a bipartite graph exists **do**
4: **/* Contention phase */**
5: $Se\_vm \leftarrow \emptyset$
6: **for** each vertex p in P on the bipartite graph **do**
7: $max\_profit \leftarrow 0$
8: **for** each vertex v in V on the bipartite graph **do**
9: $profit \leftarrow profitCalculation()$
10: /*The function $profitCalculation$ returns the corresponding profit if the VM v is placed at the PM p.*/
11: **if** $profit > max\_profit$ **then**
12: $expectedVM \leftarrow v$
13: $perchedPM \leftarrow p$
14: $max\_profit \leftarrow profit$
15: $demandedResource \leftarrow v.demandedResource$
16: **end if**
17: **end for**
18: **if** $expectedVM$ has a corresponding tuple t in Se_vm **then**
19: **if** $max\_profit > profit$ **then**
20: Replace the tuple t as ($expectedVM$, $perchedPM$, max_profit, $demandedResource$).
21: **else**
22: new a tuple $t \leftarrow$ ($expectedVM$, $perchedPM$, max_profit, $demandedResource$).
23: $Se\_vm \leftarrow Se\_vm \bigcup t$
24: **end if**
25: **end if**
26: **end for**
27: **/* Placement phase */**
28: $Sr\_vm \leftarrow \emptyset$, $Sr\_link \leftarrow \emptyset$
29: **for** each tuple t in Se_vm **do**
30: $\{v, p\} \leftarrow$ retrieve the VM and PM of t.
31: Place VM v in PM p.
32: $p.capacity \leftarrow p.capacity - v.demandedResources$
33: $Sr\_vm \leftarrow Sr\_vm \bigcup v$
34: **for** each vertex vt on the bipartite graph connected to p **do**
35: **if** $vt.demandedResource > p.capacity$ **then**
36: **end if**
37: $Sr\_link \leftarrow Sr\_link \bigcup (vt, p)$
38: **end for**
39: **end for**
40: **/*Reformation phase*/**
41: Reform the bipartite graph by removing the vertexes in Sr_vm and the links in Sr_link.
42: **end while**

Fig. 6. The QoS-aware VM placement (QAVMP) algorithm.

Theorem 2. *The proposed heuristic algorithm takes the polynomial time complexity $O(|V|^2|P|)$ to solve the QAVMP problem.*

Proof. Based on the given heuristic algorithm, it initially takes $O(|V| \cdot |P|)$ to establish a bipartite graph to model all possible placement relationships between VMs and PMs. Then, the algorithm repeats the VM placement execution for a number of rounds. In each round, the VM placement execution is divided into three phases: contention, placement, and reformation. In the contention phase, the generated profit of each possible VM placement is calculated

by traversing each link (VM–PM pair) on the bipartite graph. Based on the calculated profits, the algorithm can select the expected VM of each PM (the VM to be placed at the PM). To avoiding that two or more PMs select the same expected VM, the set Se_vm is used to resolve the conflict by comparing the generated profits. Based on the above description, the time complexity of the contention phase is dominated by the profit calculations which take $O(|V| \cdot |P|)$. In the placement phase, the set Se_vm is used to practically place VMs at their preferable PMs. It is well-known that the time complexity analysis of an algorithm is done under the worst case consideration. In the worst case, the set Se_vm only contains one item of VM placement information, such that all PMs have the same expected VM. The expected VM has $|P|$ links with all the PMs on the bipartite graph. In such case, the time complexity of the placement phase is $O(1)$. In the reformation phase, it takes $O(|P|)$ to remove the above $|P|$ links on the original bipartite graph. Under the worst case, a round can only determine the placement information of one VM. The maximum number of the rounds for the heuristic algorithm is $|V|$ since the total number of VMs to be placed is $|V|$. The entire heuristic algorithm runs in $O(|V|^2|P|)$. □

4. Performance evaluation

We developed simulation programs using MatLab [33] to evaluate the performance of our heuristic QAVMP algorithm and make comparisons with three intuitive VM placement algorithms: random-fit, first-fit, and least-fit. When a VM creation request is issued, there may have multiple PM candidates with enough resources to hold the VM. The random-fit algorithm randomly selects one of the PM candidates. The first-fit algorithm always selects the PM candidate with the smallest ID. The least-fit algorithm selects the PM candidate with the least remaining resources after placing the VM. Here, the least-fit algorithm is instead of the optimal VM placement algorithm in the previous literature. Note that all the above three intuitive algorithms do not concern the VM interference.

4.1. Simulation environment

In our simulation experiments, we assumed that the cloud computing system has 250 PMs which are randomly distributed within 10 PM groups (clusters). The Tiers model [34] is used generate the network topology of the cloud computing system. First, each of 10 PM groups (clusters) is assumed to be located within a local area over a 100 × 100 unit square plane. Among the 10 clusters, one is specified as the central cluster to organize all the 10 PM clusters as a tree topology. In each PM cluster, the PMs are assumed to be randomly deployed in the located local area of the cluster. In addition, there is a switch in each PM cluster to provide both the intra-cluster communication and the inter-cluster communication between the PMs.

Based on the system architecture, the simulation experiments were performed over the following parameter settings.

- In each PM, there have been a number of existing VMs in it. The number of the existing VM is randomly determined from 0 to 10.
- The amount of available resources is represented as a triple-tuple (available CPU GHz, available memory space in GB, available storage space in GB). The resource interval [(12, 129, 200), (96, 3000, 9600)] is used to randomly decide the available resources of each PM.
- The bandwidth is assumed to be within [10 Gbps, 40 Gbps] for the transmission line between a PM and its connected switch.
- A number of new VMs is assumed to be created within 250 PMs. The number of new VMs is set from 200 to 1000 in each simulation run, respectively.
- The amount of the resources required for a new VM was set by referring to the Amazon EC2 with different re-source demands [35].
- To simulate the profit metric, the price of a VM (the given revenue of a VM) and the penalty payment due to the QoS violation in a VM are given. We also refer to Amazon EC2 to set the price of each VM [36]. As mentioned in Section 1, the QoS requirement of an application running in a VM is dependent on the data access delay. For an application, its expected data access delay is specified within (0.5 ms, 10 ms). If the QoS requirement is violated, the penalty payment is set using the violation ratio × the price of the VM. Note that the violation ratio is defined as the ratio of the estimated data access delay to the expected data access delay specified in the SLA.

4.2. Simulation results

The following simulation results show the mean of 50 simulation runs. Fig. 7(a) shows the generated profits by creating different numbers of VMs from 200 to 1000. As seen from Fig. 7(a), the profits of all algorithms increase with the number of VMs. However, the random-fit, first-fit, least-fit algorithms do not take the VM interference into account. As creating more VMs, it correspondingly increases the effect of VM interference. The VM interference will affect the QoS requirements of applications. If the effect of the VM interference is not controlled, the QoS violation of applications will result in penalty payment to reduce the profit gain of the cloud provider. In the proposed heuristic algorithm, the VM interference is reduced as much as possible. There is a linear trend in the profit growth as increasing the number of created VMs. Compared to the three intuitive algorithms, the heuristic algorithm can enhance their profits by about 26%, 22%, and 31%, respectively.

Fig. 7(b) shows the comparison among the four algorithms in the average number of QoS-violated VMs (the VMs without satisfying QoS requirements). In the random-fit, first-fit and least-fit algorithms, we can obviously see that there is a rapid growth in the number of QoS-violated VMs. With considering the VM interference and the QoS requirements of applications, the heuristic algorithm avoids severely affecting the performance of created VMs to result in violating the QoS requirements of their running applications. Therefore, there is a smooth increase in the number of QoS-violated VMs as increasing the number of created VMs. On average, the number of QoS-violated VMs in the heuristic algorithm is about one-third of the numbers in the three intuitive algorithms.

The number of QoS-violated VMs substantially affects the penalty payment of the cloud provider. The proposed heuristic algorithm is based on the profit maximization to place VMs in PMs. For a VM i, if its placement is inevitable to violate the QoS requirements of applications running in some VMs, it will be placed at the PM with the less penalty payment to acquire the larger profit of the cloud provider. Therefore, in Fig. 7(c), we can also see that the average penalty payment in the heuristic algorithm is less than the three intuitive algorithms. The average amount of penalty payment in the heuristic algorithm is about one-third of the three intuitive algorithms.

For the number of VMs created in the PMs, the least-fit algorithm can fully exploit the resources of PMs since it attempts to place the VM at the PM with few available resources. The PMs with more available resources are used to host more VMs. Basically, the least-fit algorithm should have better performance in the number of created VMs than the other two intuitive algorithms and our proposed heuristic algorithm. In the heuristic algorithm, it also attempts to place many VMs for maximizing the profit of the cloud provider in addition to reducing the VM interference. From Fig. 7(d), we can see that the proposed heuristic algorithm is close to the least-fit algorithm in the number of VMs created. Compared

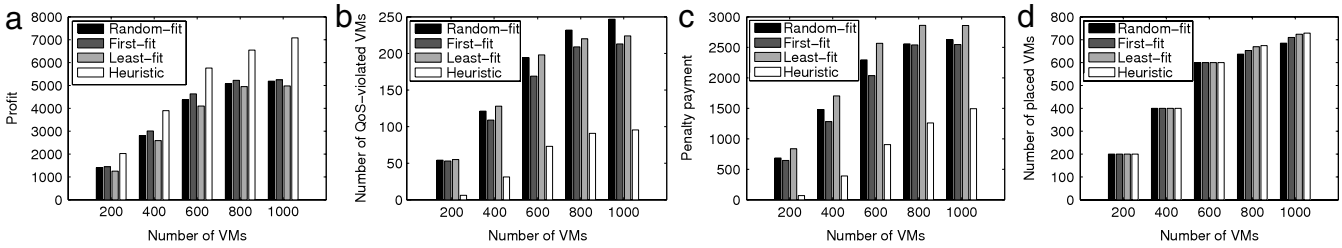

Fig. 7. Comparisons between the heuristic algorithm and three intuitive algorithms. (a) Profit. (b) Number of QoS-violated VMs. (c) Penalty payment. (d) Number of placed VMs.

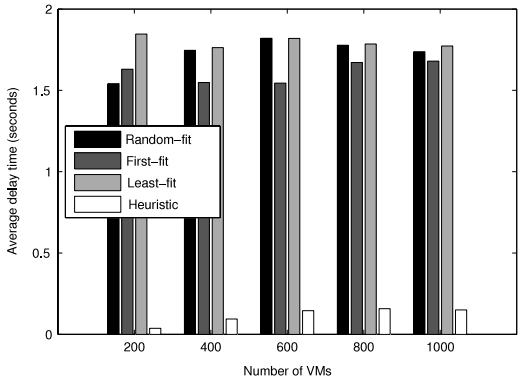

Fig. 8. Comparison of average delay time of a network I/O request.

to the two algorithms, the heuristic algorithm increases 2% of number of VMs created.

Fig. 8 depicts the average delay time of a network I/O request. We assume that a data access is issued for retrieving 1 MB data from a certain node. Due to considering VM interference, the data response (delay) time is taken into account while hosting VMs in PMs. In contrast, the data delay time is not concerned by other three algorithms in the VM placement. Therefore, the average delay time of the proposed heuristic algorithm is much less than that of the other three algorithms. As shown in Fig. 8, the proposed heuristic algorithm improves about 93% average delay time of the other three intuitive algorithms.

In Section 3, we have proved that the studied QAVMP problem is an NP-hard problem. The best VM placement will take much time. It can be only performed in a small-scale simulation environment to compare the proposed heuristic VM placement with the best VM placement. For the comparison, we assume that there are 5 PMs. The numbers of VMs to be placed are from 1 to 11. Fig. 9 illustrates the comparison results between the heuristic and best solutions. From Fig. 9(a), the profit generated by the heuristic solution is 99% of that generated by the best solution. Both the heuristic and best solutions can avoid the VM interference occurrence. Fig. 9(b) shows the comparison in the average number of QoS-violated VMs. In this metric, the heuristic solution increases about 0.07%. For the penalty payment, the best solution can reduce about 13% of the heuristic solution. In the number of placed VMs, the both solutions have the almost same value.

5. Conclusions

We have investigated the QoS-aware virtual machine placement (QAVMP) problem for efficiently performing data sensitive applications in cloud computing systems. In addition to exploiting the resources of PMs, the QoS requirements of applications and the interference among VMs are also considered in the QAVMP problem. To optimally solve the QAVMP problem, we formulate the QAVMP problem as an ILP model by integrating the three concerned factors of the QAVMP problem into the profit metric of the cloud provider. However, the ILP model involves the complicated computation. Moreover, solving ILP is an NP-complete problem [28]. To seek a time-efficient solution of the QAVMP problem, we also propose a heuristic algorithm. The time complexity of the heuristic algorithm is with the polynomial time $O(|V|^2|P|)$. In the heuristic algorithm, a bipartite graph is modeled to represent all the possible placement relationships between VMs and PMs. Then, the VMs follow a certain sequence to be placed at their preferable PMs. The VM placement sequence is designed to increase the VM placement profit of the cloud provider as much as possible. Compare to other VM placement schemes, the simulation results showed that the heuristic algorithm is superior in various performance metrics including the generated profit, the number of QoS-violated VMs, the penalty payment due to QoS violation, and the number of placed VMs.

In the future, we plan to implement the proposed QAVMP algorithm in a real cloud computing system. Moreover, the VM placement algorithm will be also extended to concern energy consumption.

Acknowledgment

This research was supported by the National Science Council, Taiwan, ROC, under Grant NSC 99-2221-E-030-007-MY3.

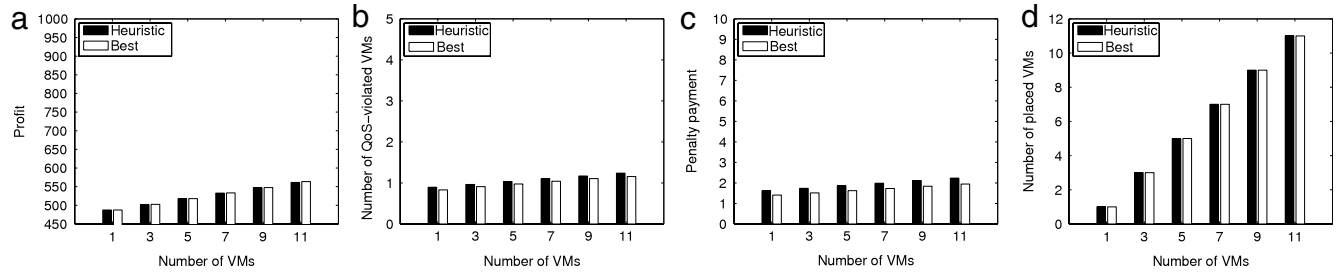

Fig. 9. Comparisons between the heuristic algorithm and the best VM placement from the ILP model. (a) Profit. (b) Number of QoS-violated VMs. (c) Penalty payment. (d) Number of placed VMs.

References

[1] Amazon EC2, 2013. URL http://aws.amazon.com/ec2/.
[2] Google Compute Engine, 2013. URL http://cloud.google.com/compute.
[3] GoGrid, 2013. URL http://www.gogrid.com/.
[4] Amazon EC2 Instance Types, 2013. URL http://aws.amazon.com/ec2/instance-types/.
[5] Xen, 2013. URL http://xen.org/.
[6] VMware, 2013. URL http://www.vmware.com/.
[7] Kernel Based Virtual Machine, 2013. URL http://www.linux-kvm.org/page/Main_Page/.
[8] S. Ibrahim, B. He, H. Jin, Towards pay-as-you-consume cloud computing, in: Proc. IEEE Int. Conf. Services Comput., 2011, pp. 370–377.
[9] B. Zhang, X. Wang, R. Lai, L. Yang, Y. Luo, X. Li, A survey on I/O virtualization and optimization, in: Proc. 5th Ann. ChinaGrid Conf., 2010, pp. 117–123.
[10] L. Wang, R.A. Hosn, C. Tang, Remediating overload in over-subscribed computing environments, in: Proc. 2012 IEEE 5th Int. Conf. Cloud Computing (CLOUD), 2012, pp. 860–867.
[11] S. Chen, J. Wu, Z. Lu, A cloud computing resource scheduling policy based on genetic algorithm with multiple fitness, in: Proc. 2012 IEEE 12th Int. Confe. Computer and Information Technology, CIT, 2012, pp. 177–184.
[12] U. Lampe, T. Mayer, J. Hiemer, D. Schuller, R. Steinmetz, Enabling cost-efficient software service distribution in infrastructure clouds at run time, in: Proc. 2011 IEEE Int. Conf. Service-Oriented Computing and Applications, SOCA, 2011, pp. 1–8.
[13] M.G. Kallitsis, R.D. Callaway, M. Devetsikiotis, G. Michailidis, Presence-aware optimum resource allocation for virtual collaboration web 3.0 environments, in: IEEE Workshop GLOBECOM '2009, 2009, pp. 1–5.
[14] P. Campegiani, F.L. Presti, A general model for virtual machines resources allocation in multi-tier distributed systems, in: Proc. 5th Int. Conf. Autonomic and Autonomous Systems, 2009, pp. 162–167.
[15] H.N. Van, F.D. Tran, J.-M. Menaud, Autonomic virtual resource management for service hosting platforms, in: Proc. Workshop Software Eng. Challenges of Cloud Comput., Vancouver, Canada, 2009, pp. 1–8.
[16] J. Xu, J.A.B. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proc. IEEE/ACM Int. Conf. Green Comput. and Commun. & Int. Conf. Cyber, Physical and Social Comput., Washington, DC, USA, 2010, pp. 179–188.
[17] S. Chaisiri, B.-S. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: IEEE Asia–Pacific Services Comput. Conf., APSCC '09, 2009, pp. 103–110.
[18] S. Chaisiri, B.-S. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, IEEE Trans. Serv. Comput. 99 (2012).
[19] K. Zamanifar, Data-aware virtual machine placement and rate allocation in cloud environment, in: Proc. 2012 2nd Int. Conf. Adv. Comput. & Commun. Technologies, ACCT, 2012, pp. 357–360.
[20] P.R. Barham, B. Dragovic, K.A. Fraser, S.M. Hand, T.L. Harris, A.C. Ho, E. Kotsovinos, A.V. Madhavapeddy, R. Neugebauer, I.A. Pratt, A.K. Warfield, Xen 2002, Tech. Rep, University of Cambridge, Cambridge, 2003.
[21] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proc. 19th ACM Symp. Operating Systems Principles, SOSP '03, 2003, pp. 164–177.
[22] Xen Cloud Platform, 2012. URL http://xen.org/products/cloudxen.html.
[23] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, Y. Cao, Who is your neighbor: net I/O performance interference in virtualized clouds, IEEE Trans. Serv. Comput. 99 (2013).
[24] D. Boutcher, A. Chandra, Does virtualization make disk scheduling passé? SIGOPS Oper. Syst. Rev. 44 (2010) 20–24.
[25] M. Kesavan, A. Gavrilovska, K. Schwan, On disk I/O scheduling in virtual machines, in: Proc. 2nd Workshop on I/O Virtualization, Pittsburgh, PA, USA, 2010, pp. 1–6.
[26] Y. Mei, L. Liu, X. Pu, S. Sivathanu, Performance measurements and analysis of network I/O applications in virtualized cloud, in: Proc. IEEE 2010 Int. Conf. on Cloud Comput. (Cloud '10), Florida, USA, 2010, pp. 59–66.
[27] D. Ongaro, A.L. Cox, S. Rixner, Scheduling I/O in virtual machine monitors, in: Proc 4th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environments, New York, USA, 2008, pp. 1–10.
[28] S. Dasgupta, C.H. Papadimitriou, U.V. Vaziran, Algorithms, McGraw-Hill, 2006.
[29] D. Gross, C.M. Harris, Fundamentals of Queueing Theory, third ed., Wiley, New York, NY, 1998.
[30] L. Li, An optimistic differentiated service job scheduling system for cloud computing service users and providers, in: Proc. 2009 3rd Int. Conf. Multimedia and Ubiquitous Engineering, 2009, pp. 295–299.
[31] A. Sodan, Predictive space- and time-resource allocation for parallel job scheduling in clusters, in: Proc. IEEE 2010 Int. Conf. Parallel Processing Workshops, 2010, pp. 313–322.
[32] K. Dutta, R. Guin, S. Banerjee, S. Chakrabarti, U. Biswas, A smart job scheduling system for cloud computing service providers and users: modeling and simulation, in: Proc. 2012 1st Int. Conf. Recent Advances in Information Technology, RAIT, 2012, pp. 346–351.
[33] MATLAB—The Language Of Technical Computing, 2010. URL http://www.mathworks.com/.
[34] K.L. Calvert, M.B. Doar, E.W. Zegura, Modeling internet topology, IEEE Commun. Soc. 35 (1997) 160–163.
[35] Amazon EC2—Instance Types, 2013. URL http://aws.amazon.com/ec2/#instance.
[36] Amazon EC2 Pricing, 2013. URL http://aws.amazon.com/ec2/pricing/.

Jenn-Wei Lin is currently a professor in the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan. He received the M.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 1993, and the Ph.D. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1999. He was a researcher at Chunghwa Telecom Co., Ltd., Taoyuan, Taiwan from 1993 to 2001. His current research interests are cloud computing, mobile computing and networks, distributed systems, and fault-tolerant computing.

Chien-Hung Chen is currently a Ph.D. student in the Department of Electrical Engineering, National Taiwan University. He received the B.S. degree in computer science and information engineering from the Chung Hua University, Taiwan, in 2008, and the M.S. degree in computer science and information engineering from Fu Jen Catholic University, Taiwan, in 2012. His research interests include cloud computing, mobile networks, and fault-tolerant computing.

Chi-Yi Lin is an associate professor in the Department of Computer Science and Information Engineering at Tamkang University, Taipei, Taiwan. He received his B.S. and Ph.D. degrees in electrical engineering from National Taiwan University in 1995 and 2003, respectively. He was a visiting researcher at AT&T Labs-Research, New Jersey from August to December, 2000, an Assistant Researcher at the Telecommunication Labs, Chunghwa Telecom from 2003 to 2007, and a Postdoctoral Research Fellow at the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology from 2007 to 2008. His research interests include cloud computing, ubiquitous computing, and social networks.