information

MDPI

Article

An RTT-Aware Virtual Machine Placement Method

Li Quan [1], Zhiliang Wang [1,*] and Fuji Ren [2]

[1] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China; quanli19871103@163.com
[2] School of Computer and Information, Hefei University of Technology, Hefei 230000, China; ren2fuji@gmail.com
* Correspondence: wzl@ustb.edu.cn; Tel.: +86-10-6233-2641

Received: 29 November 2017; Accepted: 26 December 2017; Published: 29 December 2017

Abstract: Virtualization is a key technology for mobile cloud computing (MCC) and the virtual machine (VM) is a core component of virtualization. VM provides a relatively independent running environment for different applications. Therefore, the VM placement problem focuses on how to place VMs on optimal physical machines, which ensures efficient use of resources and the quality of service, etc. Most previous work focuses on energy consumption, network traffic between VMs and so on and rarely consider the delay for end users' requests. In contrast, the latency between requests and VMs is considered in this paper for the scenario of optimal VM placement in MCC. In order to minimize average RTT for all requests, the round-trip time (RTT) is first used as the metric for the latency of requests. Based on our proposed RTT metric, an RTT-Aware VM placement algorithm is then proposed to minimize the average RTT. Furthermore, the case in which one of the core switches does not work is considered. A VM rescheduling algorithm is proposed to keep the average RTT lower and reduce the fluctuation of the average RTT. Finally, in the simulation study, our algorithm shows its advantage over existing methods, including random placement, the traffic-aware VM placement algorithm and the remaining utilization-aware algorithm.

Keywords: virtual machine placement; round-trip time; cloud computing; mobile cloud computing

1. Introduction

With the popularization of the internet of things and cloud computing technology, a large number of smart devices are used by people [1–3]. Smart devices need to leverage cloud computing to expand their capabilities, because of their limited memory, storage and CPU. There is an increasing need for services provided by the cloud. Mobile cloud computing (MCC) [4,5] is a new paradigm for addressing this issue. We can migrate some resource-intensive services to the cloud by MCC. Virtualization technology [6–8] provides a relatively independent and safe operating environment for different services in the cloud. It enables the resource provisioning efficiently that the required memory, storage and CPU resources can be packed into virtual machines (VMs). Therefore, a user's specific request could be handled by one or more VMs in the cloud. These VMs are placed on different physical machines (PMs) that may be geographically distributed and in different locations of the network topology.

The virtual machine placement problem [9] focuses on choosing optimal PMs to run needed VMs when facing lots of requests. The optimization objectives of existing researches include energy consumption, network traffic between VMs and so on and rarely consider the delay for requests. However, there are always many delay-sensitive applications among various cloud-based applications. They need assistance from cloud computing but also have minimum delay requirements, such as sensor and in real-time applications.

In order to meet the minimum delay requirements for requests, we take the round-trip time (RTT) [10] as the metric for placing VMs dynamically in a cloud. In this paper, the dynamic VM placement is studied for minimizing average RTT of all requests. First, we propose an architecture for dynamic VM placement, where users' requests are allocated to the corresponding core switches according to load balancer. The adopted network topology of physical machines is based on fat-tree [11] that is a universal network for provably efficient communication. Then, we propose a dynamic VM placement algorithm aimed at minimizing the average RTT for cloud service requests. Finally, considering the unexpected situation where a core switch is out of work due to long-running or physical damage. A dynamic VM rescheduling algorithm is introduced to keep the average RTT lower and reduce the fluctuation of RTT.

This paper is organized as follows: Section 2 describes the related researches. We adopt a network topology of physical machines deployed in the cloud, based on fat-tree, and model the problem mathematically in Section 3. In Section 4, we introduce all related algorithms in detail. Section 5 shows the advantages of our proposed algorithms through experiment. In Section 6, we summarize our paper.

2. Related Works

Numerous researchers have studied dynamic VM placement problem in cloud computing environment [12–14]. In addition, some newer computing paradigms are used for delay-sensitive tasks, such as Mobile Edge Computing, Fog Computing and Dew Computing. Luan et al. [15] outlines the main features of Fog computing which can serve mobile users with a direct short-fat connection. These newer paradigms use nearby resources to augment devices from fixed hardware [16] and other mobile devices [17]. Nevertheless, different previous studies, this paper mainly focuses on the MCC area where smart devices are augmented via remote cloud resources.

Xiong et al. [18] propose a novel VM placement policy that prefers placing a migratable VM on a host that has the minimum correlation coefficient. The correlation coefficient represents the relationship between the migrating VM and each host. A greater correlation coefficient indicates a greater performance degradation for other VMs on this host due to the migration. They focus on dynamic consolidation of VMs in the data center to reduce the energy consumption and improve physical resource utilization.

Han et al. [19] consider energy consumption and propose a remaining utilization-aware (RUA) algorithm for virtual machine (VM) placement. Besides a power-aware algorithm (PA) is proposed to find proper hosts to shut down for energy saving. Their research focuses on the multimedia cloud and tries to minimize its energy consumption on the basis of satisfying the consumers' resource requirements and guaranteeing the quality of service (QoS).

Luo et al. [20] focus on improving the overall performance of datacenters and propose a virtual machine allocation and scheduling algorithm. In order to improve the utilization of physical machines and lower energy cost, they design a self-adaptive network-aware virtual machine re-scheduling algorithm. Resource fragments and high-cost network communication VMs could be detected automatically by using the proposed algorithm and corresponding VMs are rescheduled through appropriate live migrations.

Pan et al. [21] try to address heterogeneous VMs placement optimization problem by proposing a cross-entropy-based admission control approach. In their paper, a cloud provider may offer multi-types of VMs that are associated with varying configurations and different prices. In order to fulfill users' requests, accept and place multiple VM service requests into its cloud data centers optimally to maximize the total revenue is a major problem. They model the revenue maximization problem as a multiple-dimensional knapsack problem.

Meng et al. [22] consider the scalability of cloud data centers to propose a traffic-aware virtual machine (VM) placement method. They formulate the VM placement as an optimization problem and prove its hardness. By optimizing the placement of VMs on PMs, traffic patterns among VMs

can be better aligned with the communication distance between them, e.g., VMs with large mutual bandwidth usage are assigned to PMs in close proximity.

Yapicioglu et al. [23] propose an algorithm with low computational complexity to decrease networking costs where power and communication patterns of VMs are taken into account. This paper clusters VMs according to the dynamic network traffic data and places corresponding clusters into PMs. They put frequently communicating VMs together into the same PM or as near as possible to decrease the traffic between PMs while minimizing networking delay based on average communication path and keeping the number of active servers and networking elements at a minimum to save energy.

Ilkhechi et al. [24] mainly study network and traffic aware VM placement from the perspective of a provider, where a large number of endpoints are VMs located in PMs. In this scenario, they focus on the network rather than data server constraints associated with VM placement problem. This paper proposes a nearly optimal placement algorithm that maps a set of VMs into a set of PMs with maximizing a particular metric named satisfaction.

Cohen et al. [25] focus on the network aspect and study the VM placement problem of applications with intense bandwidth requirements. The available network link may be used by many PMs and thus by the VMs placed on these PMs. They try to maximize the benefit from the overall communication sent by the VMs to a single point in a data center and propose a polynomial-time constant approximation algorithm for this problem.

Though many papers above have studied the dynamic VM placement problem, their optimization objectives manly include energy consumption, resource utilization and traffic cost between VMs etc. and rarely involve latency for requests. Many delay-sensitive applications in the MCC environment, such as sensor and in real-time applications that have minimum delay requirements. Therefore, our paper focuses on minimizing the average RTT for cloud service requests.

3. Dynamic VM Placement

3.1. Proposed Architecture

In order to place VMs dynamically according to users' requests. We propose an architecture for Dynamic VM Placement that makes up of four parts, as shown in Figure 1.

- User Request: we suppose that smart devices exploit cloud resources by request-response. Smart devices' requests require the cloud center to deploy VMs for providing appropriate service.
- Load Balancer: the load balancer could redirect the incoming requests to candidate switches according to the number of requests.
- Physical Machine: physical machines are an important part of the cloud, which provide a runtime environment for virtual machines. The physical machines are connected by fat-tree network structure.
- Dynamic Allocation of VM: this part runs the scheduling algorithms that allocate the VM dynamically for different requests to minimize the average RTT.

According to Figure 1, the users' requests are connected to the load balancer that redirects these requests to candidate core switches according to the number of requests. The core switches are on the top tier of fat-tree topology in this architecture. When one of the core switches is out of work due to long-running or physical damage, the load balancer will redirect those requests to other working core switches. Moreover, the load balancer sends the detail information of these requests to the module: Dynamic Allocation of VM. The algorithms proposed in this paper run in this module, which can place the corresponding VMs in order to minimize the average RTT for these requests. An overview of our proposed algorithms is given in Figure 2. We will discuss these algorithms in detail in Section 4.

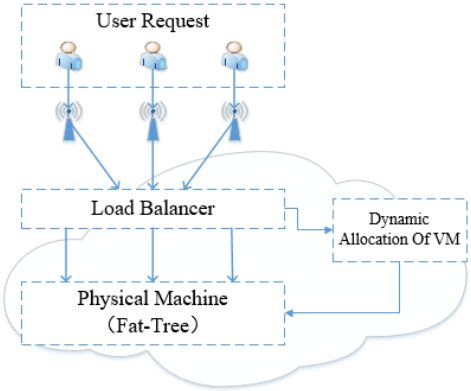


Figure 1. Architecture for Dynamic VM Placement.

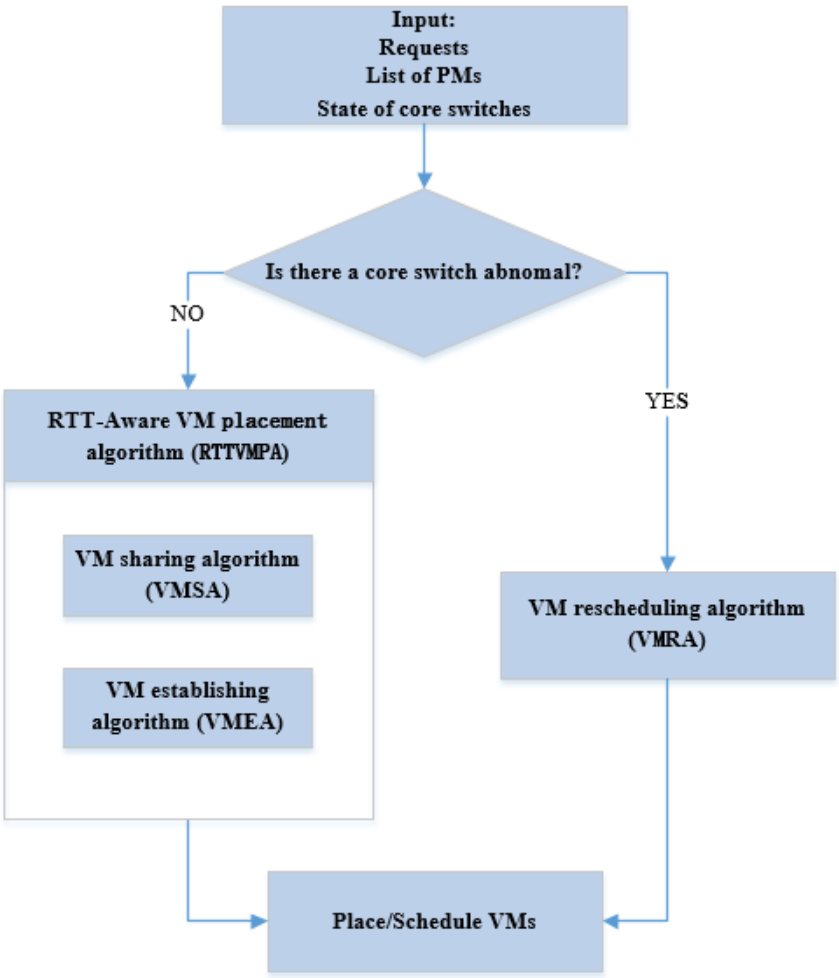


Figure 2. Overview of Proposed Algorithms.

3.2. Problem Formulation

As shown in Figure 3, it is our adopted network architecture [26] of physical machines deployed in the cloud. The architecture makes use of fat-tree topology and is composed of three tiers. Core switches (Cs_i) are on top tier of the network architecture, that deliver corresponding users' requests according to load balancer shown in Figure 1. Each point of delivery (pod) comprises with aggregation switches (Ags_j) and access switches (Acs_k), they connect with all involving core switches. These two switches perform different network functions. The aggregation switches are capable of supporting

many 10 GigE and GigE interconnects while providing a high-speed switching fabric with a high forwarding rate. They are required to provide redundancy and to maintain session state while providing valuable services to the access layer. In addition, the access switches can connect with PMs directly and provide a high GigE port density. The number of pods and three types of switches is related to the number of ports per switch. Besides, the number of ports per switch also determines the number of physical machines in pod. Suppose the number of ports per switch is p, then there will be p pods that contain $p/2$ aggregation switches and $p/2$ access switches in each pod. There are $p^2/4$ cores switches that connect to each pod with $p^2/4$ physical machines. Therefore, we can get $5p^2/4$ switches that communicate with $p^3/4$ physical machines in total.

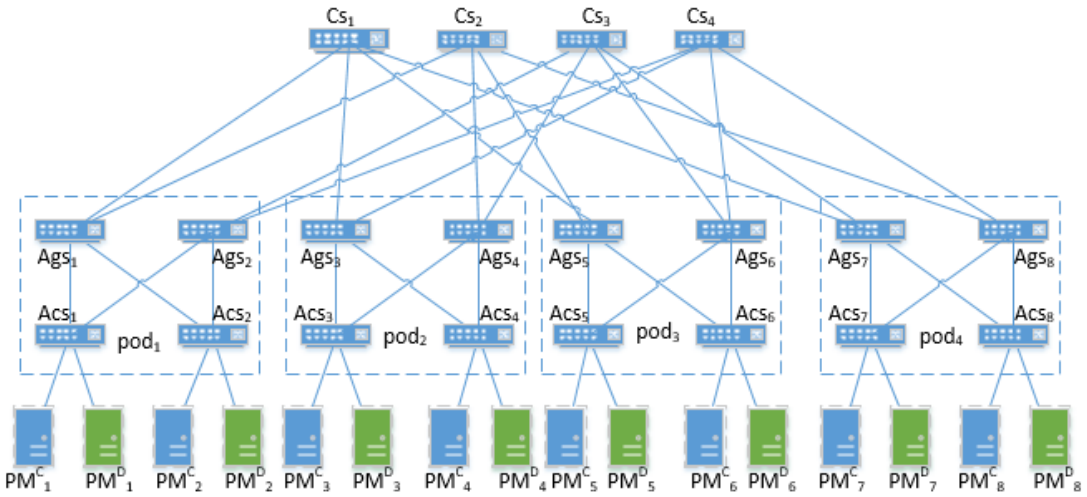


Figure 3. Network topology based on fat-tree.

In this paper, a request communicates with VMs that provides corresponding services through mentioned above three tiers topology. These PMs that VMs could be placed on are commonly distributed location, where different pods are distributed in a datacenter as shown in Figure 3. Take a specific PM PM_1^C as an example, the distance between PM_1^C and the four core switches is same. However, the switches in the communication path between them are not exactly the same. This can lead to a different request delay that affects quality of service (QoS) [27]. In order to provide low delay service especially for delay-sensitive requests, these VMs should be placed on optimal PMs to make the average delay minimization.

In this paper, there are two types of PMs and their number is the same. One (PM_l^D) is fit to VMs that provides services for data, another (PM_m^C) is fit to VMs that provides services for computing. l and m are the index of PMs. We let $PM_l^D(m_D, C_D)$ denote the attribute of PM_l^D, m_D and C_D respectively represent the spare memory and spare CPU. Similarly, let $PM_m^C(m_C, C_C)$ denote the attribute of PM_m^C, m_C and C_C represent the spare memory and spare CPU respectively. Correspondingly, there are two types of VMs. One provides service for data (VM_a^D $1 \leq a \leq A$), another provides service for computing (VM_b^C $1 \leq b \leq B$). A and B respectively represent the kinds of corresponding service. The VMs for the ath kind service of data are denoted as $VM_a^D(m_a^D, C_a^D)$, m_a^D and C_a^D respectively represent the needed memory and CPU. Similarly, $VM_b^C(m_b^C, C_b^C)$ denotes the bth kind service of computing and its needed memory and CPU are represented by m_b^C and C_b^C.

Definition 1.

$$\begin{cases} PM_l^D(m_D, C_D) - VM_a^D(m_a^D, C_a^D) > 0 & m_D - m_a^D > 0 \; and \; C_D - C_a^D > 0 \\ PM_l^D(m_D, C_D) - VM_a^D(m_a^D, C_a^D) < 0 & otherwise \end{cases} \quad (1)$$

Similarly,

$$\begin{cases} PM_m^C(m_C, C_C) - VM_b^C(m_b^C, C_b^C) > 0 & m_C - m_b^C > 0 \text{ and } C_C - C_b^C > 0 \\ PM_m^C(m_C, C_C) - VM_b^C(m_b^C, C_b^C) < 0 & otherwise \end{cases} \tag{2}$$

Therefore, a VM could be placed on the selected physical machine, only when the target physical machine has enough memory and CPU capacities:

$$PM_l^D(m_D, C_D) - VM_a^D\left(m_a^D, C_a^D\right) > 0 \tag{3}$$

or

$$PM_m^C(m_C, C_C) - VM_b^C\left(m_b^C, C_b^C\right) > 0 \tag{4}$$

In this paper, we suppose that a request could be denoted as $R_r\left(VM_a^D, VM_b^C\right)$ $(1 \leq r \leq R)$, R denotes the total number of user requests. We can see that two types of VM serve a request. In order to ensure better QoS to requests especially for delay-sensitive requests, we try to minimize the latency for all requests. This is rarely discussed in previous studies. Moreover, our paper considers the situation that a VM can serve several requests. However, a VM only serves one request in previous studies.

We only consider the general situation where a request could be served by two kinds of VMs (computing and data) in order to verify our proposed algorithms. There are also some applications that access CPU-intensive services or data-intensive services in practice. These requests could be denoted as $R_r\left(VM_b^C\right)$ for CPU-intensive services, or $R_r\left(VM_a^D\right)$ for data-intensive services. For the purpose of generality, we only consider the general situation that a request is served by VM^D and VM^C. Moreover, it is possible to allocate a data VM in a computing PM and vice versa. However, we mainly focus on the average RTT for requests and intend to verify our proposed algorithms' effectiveness for coexisting two kinds PMs. Therefore, we do not discuss this specific situation in our paper.

Remark 1. *The disk resources are easier to expand than CPU and memory. Most of the previous works [19,28,29] did not consider the disk space as well. Moreover, we only regard these resources as a constraint in this paper and they are not the main discussed contents of this paper.*

The data machines mentioned in our paper are similar to Amazon servers and we proposed VMs (VM^D and VM^C) are similar to Amazon image in which users can runs different applications. Amazon image types are defined by operating system, architecture and other parameters. However, Amazon image is used to run individualized applications for different users, which is similar to VMs (VM^D and VM^C) in our paper. They also have the specific demand for CPU and memory resources.

We assume that the types of VM^D and VM^C is A and B. It means that each type of VM^D VM_a^D $(1 \leq a \leq A)$ provides different data services. In addition, the same is to VM^C. This "Types" is determined by different applications that run in VM^D and VM^C . Take VM_a^D $(1 \leq a \leq A)$ as an example, VM_1^D and VM_2^D represent different applications.

In this paper, we regard *RTT* as the metric of delay [30] for requests based on fat-tree network topology. According to the network topology shown in Figure 3, we define the *RTT* between the ith core switch Cs_i and PMs as follows:

$$rtt_{il}^D = rtt\left(Cs_i, PM_l^D\right) \tag{5}$$

and

$$rtt_{im}^C = rtt\left(Cs_i, PM_m^C\right) \tag{6}$$

where rtt_{il}^{D} is the *RTT* between core switch Cs_i and PM_l^D, rtt_{im}^{C} is the *RTT* between core switch Cs_i and PM_m^C. Function *rtt* represents the *RTT* between two network nodes. Therefore, we can get the *RTT* matrix RTT^D and RTT^C that are updated with the latest *RTT* periodically.

$$RTT^D = \begin{bmatrix} rtt_{11}^D & rtt_{12}^D & rtt_{13}^D & \cdots & rtt_{1l}^D \\ rtt_{21}^D & rtt_{22}^D & rtt_{23}^D & \cdots & rtt_{2l}^D \\ rtt_{31}^D & rtt_{32}^D & rtt_{33}^D & \cdots & rtt_{3l}^D \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ rtt_{i1}^D & rtt_{i2}^D & rtt_{i3}^D & \cdots & rtt_{il}^D \end{bmatrix} \tag{7}$$

$$RTT^C = \begin{bmatrix} rtt_{11}^C & rtt_{12}^C & rtt_{13}^C & \cdots & rtt_{1m}^C \\ rtt_{21}^C & rtt_{22}^C & rtt_{23}^C & \cdots & rtt_{2m}^C \\ rtt_{31}^C & rtt_{32}^C & rtt_{33}^C & \cdots & rtt_{3m}^C \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ rtt_{i1}^C & rtt_{i2}^C & rtt_{i3}^C & \cdots & rtt_{im}^C \end{bmatrix} \tag{8}$$

Therefore, suppose that a request $R_r\left(VM_a^D, VM_b^C\right)$ is arranged to the ith core switch Cs_i, its needed VM_a^D and VM_b^C is placed on PM_l^D and PM_m^C respectively. The request's average *RRT* could be calculated as follow:

$$Rrtt(R_r) = \left(RTT_{i,l}^D + RTT_{i,m}^C\right)/2 \tag{9}$$

Then we can define our optimization problem to minimize the total average *RTT* for R requests:

$$Artt = min\frac{\sum_{r=1}^{R} Rrtt(R_r)}{R} = \min_{\substack{l \in L \\ m \in M}} \frac{\sum_{r=1}^{R}\left(RTT_{i,l}^D + RTT_{i,m}^C\right)/2}{R} \tag{10}$$

4. Algorithms

In this section, we will introduce the algorithms in detail according to the overview shown in Figure 2. Two VM scheduling algorithms (RTTVMPA and VMRA) are proposed for minimizing the average RTT for users' requests. Moreover, the RTT-Aware VM placement algorithm (RTTVMPA) contains two sub-algorithms that will be also discussed in this section.

First, the RTTVMPA gets the state of all *PMs* and current users' requests. RTT Matrix RTT^D and RTT^C are calculated following the Formulae (5) and (6). Then sort the PMs in ascending order according to RTT for each core switch. However, the RTT between different core switches and PMs is different, the algorithm should update the state of PMs according to steps 7 and 9. Therefore, the PMs with the smallest RTT could be selected first for those requests that are allocated to same core switch, which can reduce the RTT for users' requests. Finally, the RTTVMPA schedules the corresponding VMs according to selected index of PMs. The RTTVMPA is describe as Algorithm 1:

Different from previous studies for VM placement, which a VM serves only one request. Our paper considers the situation that a VM could serve multiple requests. Therefore, the step 16 intends to use existing VMs that could serve current request. If the existing VMs reach the maximum number that they can serve, the algorithm will deploy new VMs in step 19. For above purpose, we will introduce VM sharing algorithm (VMSA) and VM establishing algorithm (VMEA) as follows.

First, we suppose a VM could serve n requests. Let state $< PM, VM, count > (0 \leq count \leq n)$ denote the state of VM placed on PM. $count$ is the number of current serving requests. If $count = n$, the virtual machine VM cannot serve other requests. The VM sharing algorithm (VMSA) is describe as Algorithm 2:

Algorithm 1 RTT-Aware VM placement algorithm (RTTVMPA)

1. Begin
2. Initial $\boldsymbol{R}$ = 0, $\boldsymbol{Artt}$ = 0, $\boldsymbol{indexD} = \mathbf{0}$, $\boldsymbol{indexC} = \mathbf{0}$
3. Get list of $\mathbf{PM}^D$s as $\boldsymbol{ListPMD}$; $< l, PM_l^D(m_D, C_D), rtt_l > \in \boldsymbol{ListPMD},\ rtt_l = \mathbf{0},\ (\mathbf{1} \le l \le \boldsymbol{L})$
4. Get list of $\mathbf{PM}^C$s as $\boldsymbol{ListPMC}$; $< l, PM_m^C(m_C, C_C), rtt_m > \in \boldsymbol{ListPMC},\ rtt_m = \mathbf{0},\ (\mathbf{1} \le m \le \boldsymbol{M})$
5. Get the *RTT* Matrix $\boldsymbol{RTT^D}$ and $\boldsymbol{RTT^C}$
6. For each core switch $\boldsymbol{Cs_i}$ $(\mathbf{1} \le \boldsymbol{i} \le \boldsymbol{I})$ do: **step:7—22**
7. For each $\boldsymbol{PM_l^D}$ in $\boldsymbol{ListPMD}$ do: $\boldsymbol{rtt_l} = \boldsymbol{RTT_{i,l}^D}$ do: step:8
8. Sort $\boldsymbol{ListPMD}$ in ascending order by $\boldsymbol{rtt_l}$
9. For each $\boldsymbol{PM_m^C}$ in $\boldsymbol{ListPMC}$ do: $\boldsymbol{rtt_m} = \boldsymbol{RTT_{i,m}^C}$ do: **step:10**
10. Sort $\boldsymbol{ListPMC}$ in ascending order by $\boldsymbol{rtt_m}$
11. Get requests set $\boldsymbol{rs}$ belonging to $\boldsymbol{Cs_i}$
12. Get the number of requests in $\boldsymbol{rs}$, $\boldsymbol{r}$, $\boldsymbol{R} = \boldsymbol{R} + \boldsymbol{r}$
13. For each request $\boldsymbol{R_r}\left(\boldsymbol{VM_a^D}, \boldsymbol{VM_b^C}\right)$ in $\boldsymbol{rs}$ do: **step:14–22**
14. Get the $\boldsymbol{indexD}th$ $\boldsymbol{PM_l^D}$ in $\boldsymbol{ListPMD}$
15. Get the $\boldsymbol{indexC}th$ $\boldsymbol{PM_m^C}$ in $\boldsymbol{ListPMD}$
16. $\boldsymbol{Bool} \leftarrow \mathbf{VMSA}\left(\boldsymbol{VM_a^D}, \boldsymbol{PM_l^D}\right)$
17. If $\boldsymbol{Bool}$ is $\boldsymbol{false}$, $\boldsymbol{Bool} \leftarrow \mathbf{VMEA}\left(\boldsymbol{VM_a^D}, \boldsymbol{PM_l^D}\right)$
18. If $\boldsymbol{Bool}$ is $\boldsymbol{false}$, $\boldsymbol{indexD} = \boldsymbol{indexD} + \mathbf{1}$
19. $\boldsymbol{Bool} \leftarrow \mathbf{VMSA}\left(\boldsymbol{VM_b^C}, \boldsymbol{PM_m^C}\right)$
20. If $\boldsymbol{Bool}$ is $\boldsymbol{false}$, $\boldsymbol{Bool} \leftarrow \mathbf{VMEA}\left(\boldsymbol{VM_b^C}, \boldsymbol{PM_m^C}\right)$
21. If $\boldsymbol{Bool}$ is $\boldsymbol{false}$, $\boldsymbol{indexC} = \boldsymbol{indexC} + \mathbf{1}$
22. Calculate $\boldsymbol{Rrtt}(\boldsymbol{R_r})$ according to Equation (9)
23. Calculate the $\boldsymbol{Artt}$ according to Equation (10)

Algorithm 2 VM sharing algorithm (VMSA)

Input: $\boldsymbol{VM}$, $\boldsymbol{PM}$
Out: $\boldsymbol{Bool}$
1. $\boldsymbol{Bool} \leftarrow \boldsymbol{false}$
2. If $\boldsymbol{VM}$ is on $\boldsymbol{PM}$, then get state < $\boldsymbol{PM}$, $\boldsymbol{VM}$, $\boldsymbol{count}$>
3. If $\boldsymbol{count} < \boldsymbol{n}$
4. $\boldsymbol{count} = \boldsymbol{count} + \mathbf{1}$
5. Share current $\boldsymbol{VM}$
6. $\boldsymbol{Bool} \leftarrow \boldsymbol{true}$
7. End if
8. End if

According to VMSA, if $count = n$, we need establish a new VM to serve requests. The VM establishing algorithm (VMEA) is describe as Algorithm 3:

Algorithm 3 VM establishing algorithm (VMEA)

Input: $\boldsymbol{VM}$, $\boldsymbol{PM}$
Out: $\boldsymbol{Bool}$
1. $\boldsymbol{Bool} \leftarrow \boldsymbol{false}$
2. If $\boldsymbol{PM} - \boldsymbol{VM} > \mathbf{0}$ according to Equation (1) or Equation (2)
3. Establish $\boldsymbol{VM}$ on current $\boldsymbol{PM}$ and set state < $\boldsymbol{PM}$, $\boldsymbol{VM}$, $\boldsymbol{count}$>
4. $\boldsymbol{count}$**=1**
5. $\boldsymbol{Bool} \leftarrow \boldsymbol{true}$
6. End if

Remark 2. *In practice, the number of requests that can be served is dynamic so that we cannot make a quantitative analysis for the relation between "n" and average RTT. Therefore, we suppose "n" is determined by the specific applications that run in VMs. We assume the maximum number of requests that can be served is "n". We can investigate the influence of the number of requests that a VM serves on our proposed RTTVMPA.*

Furthermore, our paper considers an abnormal situation where the core switches are out of work, due to long-running or physical damage. We propose a VM rescheduling algorithm (VMRA) to ensure minimum delay in above situation.

Suppose a core switch Cs_i is abnormal, the requests connected to Cs_i are allocated to other worked core switches by load balancer according to our adopted network topology for PMs. This situation will lead to different network paths between requests and needed VMs, which will increase the average *RTT* of the overall network. Therefore, we consider rescheduling related VMs to optimal PMs so that reduce the fluctuation of *RTT* when a core switch is abnormal.

According to Equation (9), we can get:

$$min(Rrtt(R_r)) = \min_{\substack{l \in L \\ m \in M}} \frac{RTT^D_{i,l} + RTT^C_{i,m}}{2} = \min_{l \in L} \frac{RTT^D_{i,l}}{2} + \min_{m \in M} \frac{RTT^C_{i,m}}{2} \tag{11}$$

Therefore, we could get $\min_{l \in L} \frac{RTT^D_{i,l}}{2}$ and $\min_{m \in M} \frac{RTT^C_{i,m}}{2}$ respectively to make $Rrtt(R_r)$ minimization. We suppose that there are E VM^Ds and VM^Cs that serve the requests connected to the abnormal core switch previously and each VM^D and VM^C serve for t requests. In order to minimize the average *RTT* when a core switch is abnormal, we can reschedule the related VMs as follow:

$$\min_{l \in L} \frac{\sum_t RTT^D_{i,l}}{2} \tag{12}$$

and

$$\min_{m \in M} \frac{\sum_t RTT^C_{i,m}}{2} \tag{13}$$

The VM rescheduling algorithm is described detail as Algorithm 4:

Algorithm 4 VM rescheduling algorithm (VMRA)

Input: $\boldsymbol{E}$ $\boldsymbol{VM^D}$**s** and $\boldsymbol{VM^C}$s that serve the requests connected to the abnormal core switch previous

1 For each $\boldsymbol{VM^D}$, $\boldsymbol{VM^C}$ in $\boldsymbol{VM^D}$**s**, $\boldsymbol{VM^C}$s

2. Get requests set $\boldsymbol{RD}$ that served by $\boldsymbol{VM^D}$ and $\boldsymbol{RC}$ that served by $\boldsymbol{VM^C}$

3. Get the index of $\boldsymbol{PM^D}$ l to satisfy $\boldsymbol{\min_{l \in L} \frac{\sum_{RD} RTT^D_{i,l}}{2}}$

4. Get the index of $\boldsymbol{PM^C}$ m to satisfy $\boldsymbol{\min_{m \in M} \frac{\sum_{RC} RTT^C_{i,m}}{2}}$

5. Reschedule the $\boldsymbol{VM^D}$ to $\boldsymbol{PM^D_l}$

6. Reschedule the $\boldsymbol{VM^C}$ to $\boldsymbol{PM^C_m}$

7. End for

5. Experiments

In this section, we mainly focus on four different experiments. First, we compare proposed algorithm RTTVMPA with the other three VM placement algorithms. Second, we investigate the effect of the number of requests that a VM can serve and the number of pods that network topology contains on our proposed RTTVMPA separately. Finally, we verify the effectiveness of our proposed VMRA when one of the core switches is abnormal.

We use CloudSim-4.0 to simulate a cloud-computing environment. CloudSim [31,32] is an extensible cloud simulation platform developed by Melbourne University. It can simulate virtual cloud resources like VMs and cloud entities like PMs. The related basic parameters are shown in Table 1. We set the parameters of physical machines as follows. PM^D: The number of CPU's cores is two, Basic frequency of CPU is 1.2 GHz and memory capacity is 6 GB. PM^C: The number of CPU's cores is two, Basic frequency of CPU is 1.8 GHz and memory capacity is 8 GB. The resource requirements for virtual machines are as follows. VM^D: Basic frequency of CPU is 1 GH and memory capacity is 1 GB. VM^C: Basic frequency of CPU is 1 GHz and memory capacity is 2 GB. "Types of VM^D" represents the types of data services that VM^D provides. The same is to "Types of VM^C".

Table 1. The basic parameters for experiments.

Basic Parameters	Values
PM^D	CPU:2cores×2 1.2 GHz, 6 GB
PM^C	CPU:2cores×2 1.8 GHz, 8 GB
VM^D	CPU:1cores 1 GHz, 1 GB
VM^C	CPU:1cores 1 GHz, 2 GB
Types of VM^D	10
Types of VM^C	10
Number of requests that a VM serves	4

We construct the network topology with six pods, nine core switches and 54 *PMs* according to Section 3.2. Besides, we assume the number of PM^D and PM^C is the same. We simulate 200 requests and allocate these requests to nine core switches randomly. We generate the round-trip time matrix RTT^D and RTT^C randomly that follow a uniform distribution in the interval $[10, 150)$ ms.

We compare proposed algorithm RTTVMPA with the following three VM placement algorithms:

- Random placement: VMs are first placed on the available PMs that have free space for these VMs but the latency for requests is not considered.
- Traffic-aware VM placement (TAVMP) algorithm [23]: TAVMP puts frequently communicating VMs into the same PMs to decrease the traffic between VMs. Such as VM^D and VM^C that serve one request should be place in the same pod according to TAVMP.
- Remaining utilization-aware (RUA) algorithm [19]: RUA intends to place VMs on less PMs to improve resource utilization. Moreover, RUA could avoid placing VMs that have a large resource requests on the same PMs for reducing resource competition between VMs. Therefore, it can decrease the probability of PMs overloading and keep PMs' status relatively stable.

In the process of placing VMs, existing methods, such as random placement, traffic-aware VM placement algorithm and remaining utilization-aware algorithm, do not regard RTT as a condition for choosing PMs. In contrast, consider the RTT between the current request and the candidate PMs according to RTT matrix, our proposed algorithm sorts the candidate PMs in ascending order according to RTT for each core switch. The PMs with the smallest RTT could be selected first for those requests that are allocated to the same core switches, which can reduce the RTT for users' requests. According to the result of Figure 4, we can see that our proposed RTTVMPA could get the lower average RTT compared with the other three algorithms. Moreover, RTTVMPA can keep a slow growth trend for average RTT when the number of requests increases and the other three algorithms obtain a higher average RTT.

We investigate the influence of the number of requests that a VM serves on our proposed RTTVMPA. In this experiment, the number of requests that a VM serves vary from 4 to 8, other settings are same to Table 1. It is shown that more number of requests that a VM serves can lead to a lower average RTT for all requests in Figure 5.

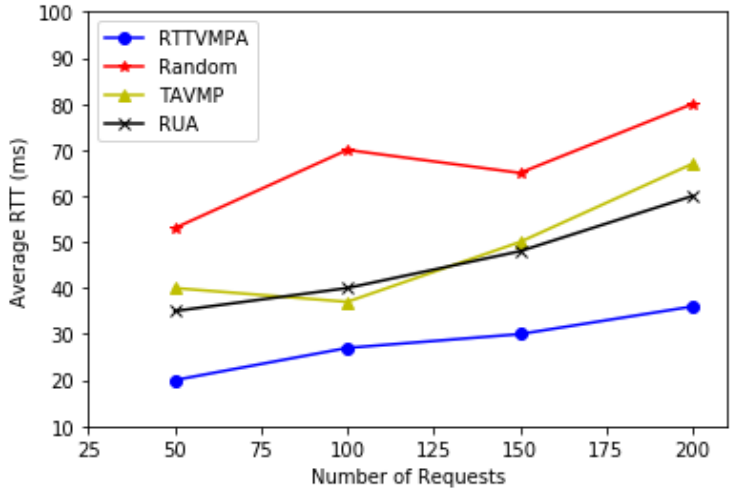


Figure 4. Shows the comparison of the four algorithms in different number of requests.

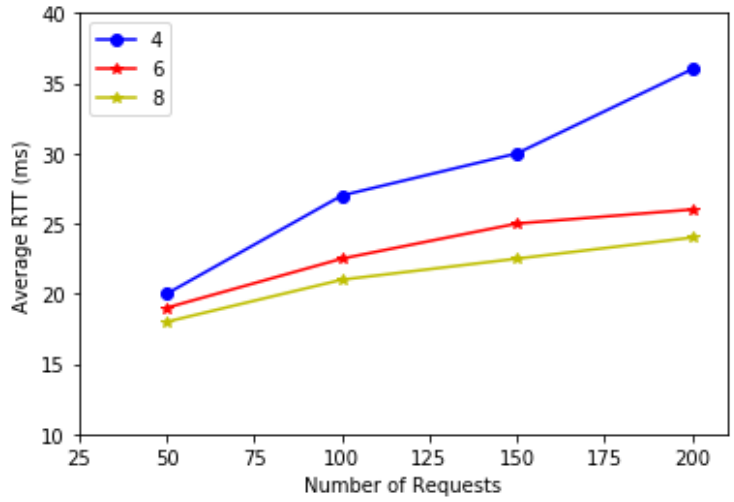


Figure 5. Number of requests that a VM serves affect average RTT.

We investigate the influence of the number of pods in our adopted network topology on our proposed RTTVMPA. Varying the number of pods from 6 to 10, other settings are same to Table 1. As shown in Figure 6, we can notice that more pods could keep lower average RTT for the same number of requests.

In order to verify our proposed VMRA, we simulate the case where one of the core switches is abnormal and the load balancer can assign corresponding requests to other working core switches. In this experiment, one of the core switches is abnormal randomly when the number of requests is 75 and 150. Therefore, re-allocation of these requests leads to the change of average RTT that can be seen when the data points are 75 and 150 as shown in Figure 7. We mark these key points in order to show these changes obviously. The proposed VMRA algorithm can reschedule the involving VMs to optimal PMs in order to decrease the fluctuation of average RTT. This situation is denoted as A_RTTVMPA_VMRA and the other situation with no VMRA is denoted as A_RTTVMPA. Then, we compare the average RTT of all requests in the above two cases. The normal situation is considered as a benchmark of comparison where all core switches are working normally. Moreover, the normal situation is denoted as N_RTTVMPA. Finally, we observe the change of the average RTT under the three different situations as mentioned above.

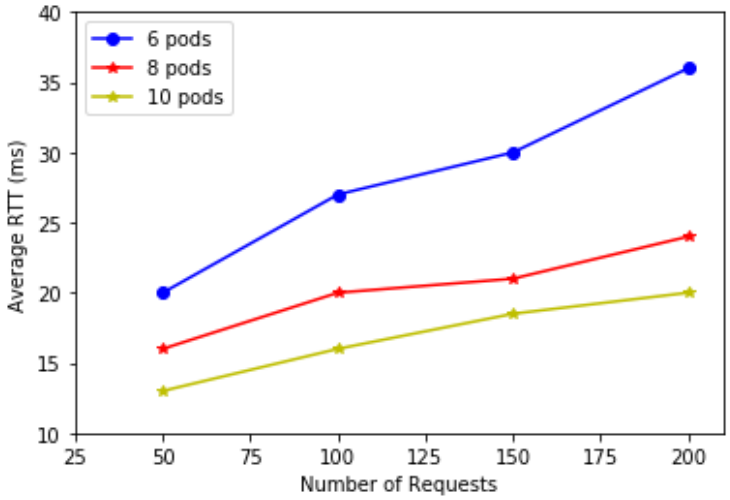
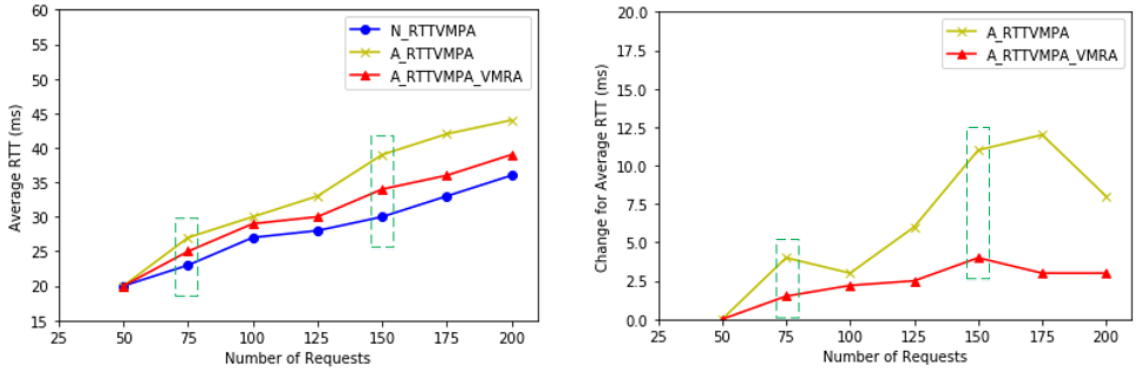


Figure 6. Number of pods affect average RTT.



(**a**) (**b**)

Figure 7. (**a**) shows the trend of average RTT for three different conditions; (**b**) shows the fluctuation of average RTT for A-RTTVMPA and A-RTTVMPA-VMRA compared with N-RTTVMPA.

The results of this experiment are shown in Figure 7, A_RTTVMPA_VMRA is better than A_RTTVMPA when the core switch is abnormal. When the number of requests is 75 and 150, the increase of average RTT is obvious for A_RTTVMPA. Because those requests that connect to the abnormal core switch are assigned to another working core switch. In order to keep a lower average RTT, our proposed VMRA tries to reschedule corresponding VMs according to changes of core switches. The A_RTTVMPA_VMRA can obtain a lower average RTT than A_RTTVMPA by using VMRA to reschedule the related VMs to optimal PMs that satisfy the Formula (11). However, the A_RTTVMPA keeps those corresponding VMs on the original PMs, which leads to a higher average RTT.

We use the difference between two methods and benchmark to show the change of average RTT when a core switch is abnormal and the comparable result is shown in Figure 7b. These changes are calculated by the following formula. The change of average RTT for A_RTTVMPA shown by a yellow line:

$$Artt(A\_RTTVMPA) - Artt(N\_RTTVMPA) \tag{14}$$

In addition, the change of average RTT for A_RTTVMPA_VMRA shown by a red line:

$$Artt(A\_RTTVMPA\_VMRA) - Artt(N\_RTTVMPA) \tag{15}$$

As shown in Figure 7b, it is obvious that RTTVMPA with VMRA could get smaller change for average RTT than RTTVMPA when a core switch is abnormal. We use the change of average RTT to show the fluctuation of average RTT for two different methods, due to a core switch being abnormal. Obviously, the average RTT can obtain a smooth fluctuation by using A_RTTVMPA_VMRA. However, the A_RTTVMPA is the opposite. The comparisons for the two methods are listed in Table 2. Therefore, we can conclude that the RTTVMPA with VMRA can keep average RTT lower and reduce the fluctuation when facing to an abnormal situation.

Table 2. The comparisons for different methods.

Methods	Average RTT	Fluctuation of Average RTT
A_RTTVMPA	Lower	Smooth
A_RTTVMPA_VMRA	Higher	Unsmooth

6. Conclusions

In this paper, we mainly study an optimal virtual machine placement method for minimizing the average RTT for users' requests. This is important for delay-sensitive applications, such as sensor and in real-time applications. Considering the RTT between the current requests and the candidate PMs, we propose an RTT-Aware VM placement algorithm. Our proposed algorithm tries to place VMs on the PMs that have the lower RTT from requests, by sorting the candidate PMs in ascending order according to RTT for each core switch. This can obtain lower average RTT for requests comparing with Random placement, traffic-aware VM placement algorithm and remaining utilization-aware algorithm. We also propose a VM rescheduling algorithm that can keep average RTT lower and reduce the fluctuation of average RTT when a core switch is abnormal. In future studies, we will consider the effective utilization of cloud resources in addition to the delay for users' requests. Moreover, reducing the packet loss rate and increasing the throughput of communication networks are also important requirements for users' requests.

Acknowledgments: This paper is supported by National Natural Science Foundation of China (Key Project, No. 61432004): Mental Health Cognition and Computing Based on Emotional Interactions. National Key Research and Development Program of China (No. 2016YFB1001404): Multimodal Data Interaction Intention Understanding in Cloud Fusion. National Key Research and Development Program of China (No. 2017YFB1002804) and National Key Research and Development Program of China (No. 2017YFB1401200).

Author Contributions: The work presented in this paper represents a collaborative effort by all authors, whereas Li Quan wrote the main paper. Zhiliang Wang and Fuji Ren discussed the proposed algorithm and comparison of the experiment. All the authors have read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Santamaria, A.F.; Serianni, A.; Raimondo, P.; De Rango, F.; Froio, M. Smart wearable device for health monitoring in the Internet of Things (IoT) domain. In Proceedings of the Summer Computer Simulation Conference, Montreal, QC, Canada, 24–27 July 2016; p. 36.
2. Majeed, A. Internet of things (IoT): A verification framework. In Proceedings of the Computing and Communication Workshop and Conference, Las Vegas, NV, USA, 9–11 January 2017; pp. 1–3.
3. Perumal, T.; Datta, S.K.; Bonnet, C. IoT device management framework for smart home scenarios. In Proceedings of the 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE), Osaka, Japan, 27–30 October 2015.
4. Dinh, H.T.; Lee, C.; Niyato, D.; Wang, P. A survey of mobile cloud computing: Architecture, applications and approaches. *Wirel. Commun. Mob. Comput.* **2013**, *13*, 1587–1611. [CrossRef]
5. Gai, K.; Qiu, M.; Zhao, H.; Tao, L.; Zong, Z. Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing. *J. Netw. Comput. Appl.* **2016**, *59*, 46–54. [CrossRef]

6. Morabito, R.; Beijar, N. Enabling data processing at the network edge through lightweight virtualization technologies. In Proceedings of the IEEE International Conference on Sensing, Communication and Networking, London, UK, 27–30 June 2016.
7. Muller, A.; Wilson, S. *Virtualization with Vmware Esx Server*; Syngress Publishing: Rockland, MA, USA, 2005.
8. Lamourine, M. Openstack. *Login Mag. USENIX SAGE* **2014**, *39*, 17–20.
9. Shiva, P.S.M.; Venkatesh, R.R.; Rolia, J.; Islam, M. Virtual Machine Placement. U.S. Patent 9,407,514, 2 August 2016.
10. Kim, D.; Lee, J. End-to-end one-way delay estimation using one-way delay variation and round-trip time. In Proceedings of the Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness & Workshops, Vancouver, BC, Canada, 14–17 August 2007; pp. 1–8.
11. Leiserson, C.E. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* **2012**, *C-34*, 892–901. [CrossRef]
12. Usmani, Z.; Singh, S. A survey of virtual machine placement techniques in a cloud data center. *Procedia Comput. Sci.* **2016**, *78*, 491–498. [CrossRef]
13. Zhan, Z.H.; Liu, X.F.; Gong, Y.J.; Zhang, J.; Chung, S.H.; Li, Y. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.* **2015**, *47*, 1–33. [CrossRef]
14. Pacini, E.; Mateos, C.; Garino, C.G. Distributed job scheduling based on swarm intelligence: A survey. *Comput. Electr. Eng.* **2014**, *40*, 252–269. [CrossRef]
15. Luan, T.H.; Gao, L.; Li, Z.; Xiang, Y.; Sun, L. Fog computing: Focusing on mobile users at the edge. *arXiv* **2015**, arXiv:1502.01815.
16. Satyanarayanan, M.; Bahl, P.; Cáceres, R.; Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Comput.* **2009**, *8*, 14–23. [CrossRef]
17. Hirsch, M.; Rodriguez, J.M.; Zunino, A.; Mateos, C. Battery-aware centralized schedulers for CPU-bound jobs in mobile Grids. *Pervasive Mob. Comput.* **2016**, *29*, 73–94. [CrossRef]
18. Fu, X.; Zhou, C. Virtual machine selection and placement for dynamic consolidation in cloud computing environment. *Front. Comput. Sci.* **2015**, *9*, 322–330. [CrossRef]
19. Han, G.; Que, W.; Jia, G.; Shu, L. An efficient virtual machine consolidation scheme for multimedia cloud computing. *Sensors* **2016**, *16*, 246. [CrossRef] [PubMed]
20. Luo, G.; Qian, Z.; Dong, M.; Ota, K.; Lu, S. Network-aware re-scheduling: Towards improving network performance of virtual machines in a data center. In Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Dalian, China, 24–27 August 2014; pp. 255–269.
21. Pan, L.; Wang, D. A cross-entropy-based admission control optimization approach for heterogeneous virtual machine placement in public clouds. *Entropy* **2016**, *18*, 95. [CrossRef]
22. Meng, X.; Pappas, V.; Zhang, L. Improving the scalability of data center networks with traffic-aware virtual machine placement. In Proceedings of the 2010 IEEE Conference on Computer Communications (INFOCOM), San Diego, CA, USA, 15–19 March 2010; pp. 1–9.
23. Yapicioglu, T.; Oktug, S. A traffic-aware virtual machine placement method for cloud data centers. In Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing, London, UK, 8–11 December 2014; pp. 299–301.
24. Ilkhechi, A.R.; Korpeoglu, I. *Network-Aware Virtual Machine Placement in Cloud Data Centers with Multiple Traffic-Intensive Components*; Elsevier North-Holland, Inc.: Duivendrecht, The Netherlands, 2015; pp. 508–527.
25. Cohen, R.; Lewin-Eytan, L.; Naor, J.; Raz, D. Almost optimal virtual machine placement for traffic intense data centers. In Proceedings of the 2013 IEEE Conference on Computer Communications (INFOCOM), Turin, Italy, 14–19 April 2013; Volume 12, pp. 355–359.
26. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *ACM Sigcomm Comput. Commun. Rev.* **2008**, *38*, 63–74. [CrossRef]
27. Pedersen, J.M.; Tahir Riaz, M.; Dubalski, B.; Ledzinski, D.; Júnior, J.C.; Patel, A. Using latency as a QoS indicator for global cloud computing services. *Concurr. Comput. Pract. Exp.* **2014**, *25*, 2488–2500. [CrossRef]
28. Lim, J.B.; Yu, H.C.; Gil, J.M.; Lim, J.B.; Yu, H.C.; Gil, J.M. An efficient and energy-aware cloud consolidation algorithm for multimedia big data applications. *Symmetry* **2017**, *9*, 184. [CrossRef]
29. Tang, Y.; Hu, Y.; Zhang, L. A classification-based virtual machine placement algorithm in mobile cloud computing. *KSII Trans. Internet Inf. Syst.* **2016**, *10*, 1998–2014.

30. Keller, M.; Karl, H. Response time-optimized distributed cloud resource allocation. In Proceedings of the 2014 ACM SIGCOMM Workshop on Distributed Cloud Computing, Chicago, IL, USA, 17–22 August 2016; pp. 47–52.
31. Calheiros, R.N.; Ranjan, R.; Beloglazov, A.; Rose, C.A.F.D.; Buyya, R. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **2011**, *41*, 23–50. [CrossRef]
32. Long, W.; Lan, Y.; Xia, Q. Using cloudsim to model and simulate cloud computing environment. In Proceedings of the International Conference on Computational Intelligence and Security, Mount Emei, China, 14–15 December 2013; pp. 323–328.