Minimizing Response Latency via Efficient Virtual Machine Placement in Cloud Systems

Hou Deng*, Liusheng Huang†, Chenkai Yang*, Hongli Xu†, Bing Leng*
School of CS & Tech., Univ. of Science & Technology of China, Hefei, Anhui 230027, P.R. China
Suzhou Institute for Advanced Study, Univ. of Science & Technology of China, Suzhou, Jiangsu 215123, P.R. China
Email: *{dengh, ckyang, lengb}@mail.ustc.edu.cn, †{lshuang, xuhongli}@ustc.edu.cn

***Abstract*—As more and more applications migrate into clouds, the placement of virtual machines for these applications has much impact on the performance of cloud systems. A number of virtual machine (VM) placement techniques have been proposed over recent years. However, most of the existing works on VM placement ignore the response latency of the requests from tenants. In this paper, we investigate the techniques of VM placement with stochastic requests from the tenants to minimize the total (average) response latency. We first model the requests for each application from the corresponding tenant as independent Poisson stream. Moreover, the VMs are modeled as simple M/M/1 queueing systems. Then, we define the problem of VM placement for minimizing the total response delay (VMMD) and show it is NP-hard. We propose three heuristic algorithms, namely, Greedy, Local Adjustment (LA) and Simulated Annealing (SA). We conduct abundant simulation experiments to evaluate the performance of our proposed algorithms. The simulation results show that the proposed algorithms are efficient in decreasing the total response latency of the requests from tenants. Especially, the SA heuristic, which decreases the total response latency about 68% at most, shows the best performance on minimizing the total response latency in cloud systems.**

***Keywords*—Cloud Systems, Virtual Machine Placement, Stochastic Requests, M/M/1 Queueing System**

I. Introduction

Cloud computing has emerged in recent years as one of the most interesting developments in technology. The adoption and deployment of cloud computing platforms have many attractive benefits, such as reliability and robustness [10]. As a result, more and more applications are migrating into clouds, and it becomes important for the cloud provider to solve the problem of how to distribute cloud resources efficiently. In modern cloud data centers, e.g. Amazon EC2 [4] and Cisco data center [5], virtual machine (VM) placement is the primary issue facing the effective scheduling of cloud resources [6]. A good placement will lead to better power efficiency and less cost (i.e. bandwidth, latency etc). A number of VM placement techniques have been proposed in the last several years. Xin *et al*. [11] have introduced a new model, which can be used to guide the design of the resource-balanced VM placement algorithm, and proposed an energy efficient VM placement algorithm EAGLE, which can save about 15% more energy. O. Biran *et al*. [12] have dealt with the problem of VM placement in the context of bandwidth usage and network constraints. Among all these works, VM placement has played an important role in the resource distribution of cloud system.

As the number of cloud applications increases, there are many popular cloud applications, such as data-intensive applications [1], which are service for dealing with the requests from tenants by data access in cloud systems. Actually, the number of request from tenant will be large. For example, in Amazon EC2 [4], the amount of requests is tremendous and growing every day. What's more, the service rate of VM is restricted in data-intensive applications. However, all these works talked above ignore the amount of requests and assume the requests will be processed once they arrive at VMs. Imposing these assumptions leads to vast waiting time of requests in the queue. On the other hand, the packet drop rate will be high when the queue is full. These disadvantages will result in inefficient, unreliability and poor user experiences. In our research, we take the requests from tenants into account and model it as stochastic quantities. The request is typically an aggregation of many independent traffic sources, and by the central limit theorem, it can be approximated using a Poisson random variable [13]. on the other hand, the service rate of VM is restricted. A request goes to the VM who holds the corresponding application and will be waiting for service in a queue. Therefore, the VMs could be modeled as simple M/M/1 queueing systems.

Moreover, the service rate of each VM is decided by two parts: the rate of processing and the rate of data transmission between VM and data node when computation and data are spread over a large number of nodes in cloud systems [1]. Since data-intensive applications are mainly accessing data between VMs and data nodes, we ignore the rate of processing. Thus, the service rate of VM is measured by the delay of data transmission between VM and data node. In this case, the service rate is relevant with the placement of VMs and nodes that store the corresponding data. For instance, if the computation VMs and corresponding data nodes are placed on different racks then the delay of data transmission will be larger than that placed on the same rack, and the corresponding service rate will be slower. Ideally, it would be preferable to keep all data access local and systems such as Hadoop [7] try to accomplish this by reducing the amount of remote data access. In general, it will not be possible to keep most data accesses local. So it is important to carefully place computation nodes (VMs) so that the service rate is larger and

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

the response latency will be minimized. Note that the response delay for the request should include the link delay and waiting time at a VM.

In this paper, we investigate the techniques of VM placement with stochastic requests from tenants to minimize the total response latency. We first model the requests for each application as independent Poisson stream, and the VMs are modeled as simple M/M/1 queueing systems. Then, we define the problem of VM placement for minimizing the total (average) response delay (VMMD) and propose three heuristic algorithms. Our contributions in this paper are listed as follows:

1. We define the problem of VM placement with stochastic requests to minimize the total (average) response delay (VMMD). It is proved that the VMMD problem is NP-hard. To our best knowledge, this is the first work that deals with the problem of VM placement in cloud systems with stochastic requests.
2. As NP-hardness, we propose three heuristic algorithms including Greedy, Local Adjustment (LA) and Simulated Annealing (SA) to solve this problem. The Greedy heuristic finds VM placement one by one with the minimum delay objective. In order to investigate a better solution, LA heuristic and SA heuristic are proposed based on Greedy heuristic.
3. The simulation results show that the proposed algorithms are efficient in decreasing the total response delay of the requests. Especially, the SA heuristic decreases the total response latency about 68% compared with Random algorithm.

The rest of the paper is organized as follows: In Section II, we discuss the related works. Section III describes the system model and problem formulation. The detailed algorithm descriptions of Greedy, Local Adjustment (LA) and Simulated Annealing (SA) are given in Section IV. The simulation results are illustrated in Section V. We conclude the paper in Section VI.

II. Related Work

In this section, we briefly review the related works about VM placement in cloud systems with two kinds of optimization goal: better power efficiency and less cost.

As energy consumption increasing in cloud data centers, many research efforts focus on making the cloud systems more energy efficient in recently years. One of the technologies is VM placement [11], [15], [16], [17], [18], [20]. For example, Gao *et al*. [15] proposed a multi-objective ant colony system algorithm for the virtual machine placement problem. Their goal is to efficiently obtain a set of non-dominated solutions (the Pareto set) that simultaneously minimize total resource wastage and power consumption. In [16], the authors proposed VMPlanner, a network-wide power manager that optimizes virtual machine (VM) placement and traffic flow routing to reduce data center power costs by sleep scheduling of network elements. Dong *et al*. [17] proposed a VM placement scheme meeting multiple resource constraints, such as the physical server size and network link capacity to improve resource utilization and reduce both the number of active physical servers and network elements so as to finally reduce energy consumption.

For the optimization goal of less cost, there also exists many studies [1], [8], [9], [14], [18], [19]. Our work is also belong to less cost (latency). Xin *et al*. [8] have formulated the VM placement problem for cost minimization, in which both physical machines (PM-cost) and network traffics (N-cost) are taken into account. In [14], the authors proposed a policy to place the VM with consideration of network conditions to minimize the data transfer time consumption and maintain application performance. To achieve better performance, live migration is adopted in their work. Vu *et al*. [18] proposed a virtual machine placement mechanism that considers traffic as well as power among VMs within a cloud data center. The goal of this paper is to minimize the communication cost and also save energy. In [19], the authors first modeled a VM placement problem for the total completion time minimization by adopting VM migration, and proposed a migration algorithm that is a heuristic approach.

However, the number of requests from tenants will be large and the service rate of VM is restricted in practice. This leads to a lot of waiting time of requests in queue. All these works above ignore the requests from tenants. These results are unrealistic in many practical placement.

III. Problem Description

In this section, we present the system model and problem formulation. We first introduce the system model. Then, we formally define the VMMD problem, based on which, the NP-hardness will be proved at last.

A. System Model

There is a set of tenants denoted by $\{A_1, A_2, \cdots, A_l\}$. Each of them need to migrate one data-intensive application into cloud system. In order to provide services to tenants, it is necessary for cloud provider to deploy all applications on the available VMs in the cloud. We assume there are two types of resources: available compute nodes (VMs) and data nodes (DNs) in a cloud data center [1]. Let $\{C_1, C_2, \cdots, C_m\}, m \geq l$ be the set of available VMs and $\{D_1, D_2, \cdots, D_n\}, n \geq l$ be the set of available DNs. In addition, we assume one VM accesses data from one DN. For example, in Amazon EC2 environment the compute node mounts the data node as a volume and accesses the volume [1]. We assume one application is processed by one VM. Moreover, our algorithms can also handle one application processed by multiple VMs by decomposing multiple instance of application.

For the sake of readability, we define two key terminologies of our proposed work.

Definition 1: A Resource Pair is denoted by a two tuple $< j, k >$, where j represents the VM C_j, and k is the DN D_k.

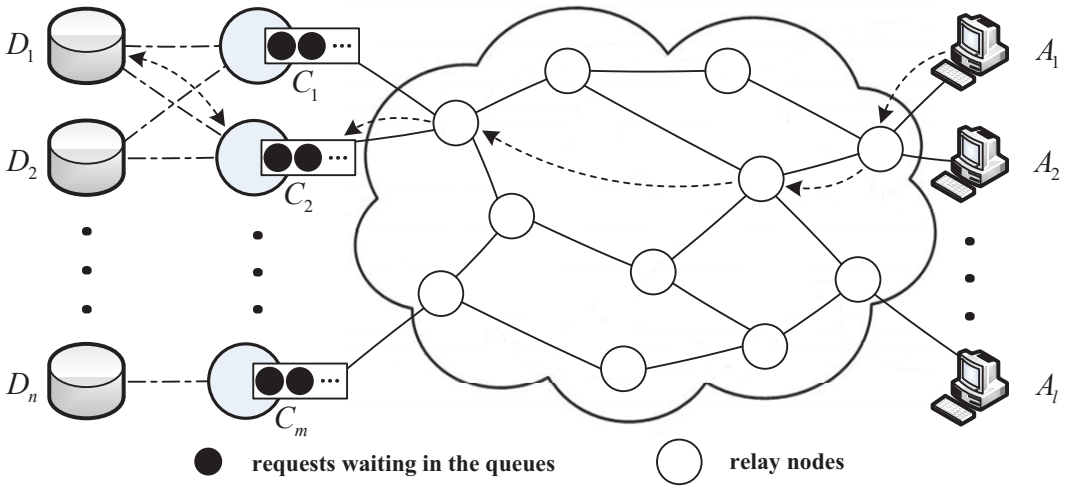

Fig. 1: The M/M/1 Queueing System for the Requests from Tenants

TABLE I: Primary Notations

Symbols	Descriptions
A	Set of the tenants ($A = \{A_1, A_2, \cdots, A_l\}$)
C	Set of the VMs ($C = \{C_1, C_2, \cdots, C_m\}$)
D	Set of the DNs ($D = \{D_1, D_2, \cdots, D_n\}$)
d_{ij}	The delay from A_i to C_j
t_{jk}	The delay between C_j and D_k
λ_i	Request rate from A_i
μ_{ijk}	Service rate for the tenant assignment $< i, j, k >$
$\overline{W}$	Upper bound for the expected waiting time of requests
α_i	The average data transfer volume corresponding to A_i

Based on the system model above, we could assign one resource pair to one tenant.

Definition 2: A Tenant Assignment is denoted by a three tuple $< i, j, k >$, where i represents the tenant A_i, and $< j, k >$ is a resource pair.

The link delay of the unit of data transmission between VM C_j and DN D_k is denoted by t_{jk}, and the link delay from tenant A_i to VM C_j is denoted by d_{ij}. Our algorithms can also handle it in geo-distributed clouds by setting the matrix of d_{ij}. According to the statements in [13], requests could be approximated using a Poisson random variable. We assume that requests from A_i form an independent Poisson stream with parameter λ_i. Moreover, the service rate of each VM is limited and decided by data access between VM and DN. The VMs are modeled as simple M/M/1 queueing systems, that is, for each tenant assignment $< i, j, k >$, the service time, which is the time of data access between C_j and D_k, at VM C_j obeys the exponential distribution with service rate μ_{ijk}. We should notice that the service rate μ_{ijk} is inversely proportional to the latency t_{jk} and the average data transfer volume α_i. For simplicity, we assume $\mu_{ijk} = \frac{1}{\alpha_i t_{jk}}$. To guarantee that each request is served in time, there is an upper bound $\overline{W}$ for the expected waiting time of requests. The primary notations used in this paper are summarized in Table I.

The M/M/1 queueing system for the requests from tenants is illustrated in Fig.1. We assume C_2 and D_1 are assigned to A_1. The request from A_1 is transmitted to C_2 via the relay nodes. Because of the limitation of processing capability of C_2, the request may not be processed immediately, but pushed into the waiting queue. When processing it, the VM C_2 will access data with D_1. Thus, the response delay for a request consists of the link delay and the waiting time in cloud system.

According to the above model, the problem of VM placement for minimizing the total response delay (VMMD) can be stated as follows: given a tenant set with request rates denoted by λ_i and the average data transfer volumes α_i, an available VM set and an available DN set with the link delay of unit data transmission between VMs to DNs denoted by t_{ij}, and an upper bound for the expected waiting time of requests $\overline{W}$, find a solution of VM placement that minimizes the total (average) delay in the cloud system.

B. Problem Formulation

In this part, we will formulate the VMMD problem. A binary three-dimensional matrix $x = [x_{ijk}]$ is introduced for it, which is assumed as:

$$x_{ijk} = \begin{cases} 1, & \text{if the resource } C_j \text{ and } D_k \text{ is assigned to } A_i, \\ 0, & \text{otherwise,} \end{cases}$$

where $\forall A_i \in A$, $\forall C_j \in C$, $\forall D_k \in D$.

In order to get the optimization objective, the response delay, which is the period of time from the beginning of a request to the end of its processing, can be divided into two parts: the travel time and waiting time. The aggregate travel time of requests from the tenant A_i per unit time is

$$P_i = \sum_{j:C_j \in C} \sum_{k:D_k \in D} \lambda_i d_{ij} x_{ijk}.$$

Since each VM behaves as an M/M/1 queueing system, when the resource C_j and D_k is assigned to tenant A_i, the expected waiting time at C_j is $W_{ijk} = 1/(\frac{1}{\alpha_i t_{jk}} - \lambda_i)$ [2]. Thus, the aggregate waiting time of requests from the tenant A_i per unit

time is

$$Q_i = \sum_{j:C_j \in C} \sum_{k:D_k \in N} \lambda_i x_{ijk} W_{ijk}.$$

Thus, The sum of the response time of requests can be formalized by:

$$S = \sum_{i:A_i \in A} (P_i + Q_i) = \sum_{i:A_i \in A} \sum_{j:C_j \in C} \sum_{k:D_k \in D} x_{ijk} d'_{ijk}.$$

where $d'_{ijk} = \lambda_i(d_{ij} + W_{ijk})$ represents the aggregate response time of requests of the tenant assignment $< i, j, k >$.

Based on the analysis above, we can obtain the following mathematical programming formulation:

$$\min S = \sum_{i:A_i \in A} \sum_{j:C_j \in C} \sum_{k:D_k \in D} x_{ijk} d'_{ijk},$$

subject to

$$\sum_{j:C_j \in C} \sum_{k:D_k \in D} x_{ijk} = 1, \qquad \forall i : A_i \in A, \quad (1)$$

$$\sum_{i:A_i \in A} \sum_{k:D_k \in D} x_{ijk} \leq 1, \qquad \forall j : C_j \in C, \quad (2)$$

$$\sum_{i:A_i \in A} \sum_{j:C_j \in C} x_{ijk} \leq 1, \qquad \forall k : D_k \in D, \quad (3)$$

$$\sum_{j:C_j \in C} \sum_{k:D_k \in D} (\lambda_i - \frac{1}{\alpha_i t_{jk}} + \frac{1}{W}) x_{ijk} \leq 0, \quad \forall i : A_i \in A, \quad (4)$$

$$x_{ijk} \in \{0, 1\}, \qquad \forall i : A_i \in A, \forall j : C_j \in C, \forall k : D_k \in D. \quad (5)$$

The objective is to minimize the total response delay of requests, subject to constraints $(1) \sim (5)$. Equation (1) ensures that each application is deployed. Inequalities (2) and (3) assure that VM and DN can be assigned only one application. Constraint (4) ensures that the expected waiting time at any VM does not exceed $\overline{W}$. Finally, the last constraint (5) ensures the binary nature of the variable x.

As shown above, the main factors that lead to the hardness of this problem include the nonlinear objective function, the large number of constraints, and the binary nature of the decision variables. It is proved that the VMMD problem defined above is NP-hard in the following theorem.

Theorem 1: The VMMD problem (defined above) is an NP-hard problem.

Proof: We assume the number of tenants, VMs and DNs are the same, that is, $l = m = n$, and the upper bound for the expected waiting time of requests $\overline{W}$ is infinity. In this case, all of the resources including VMs and DNs are assigned to tenants. Thus, the constraints (2), (3) will be equal. The constraint (4) can be ignored. The original problem is reduced to axial 3-dimensional assignment problem (3-DAP), and Karp [21] showed that the axial 3-DAP is NP-hard. So the VMMD is an NP-hard problem. ■

IV. Algorithms for The VMMD Problem

Due to NP-hardness, finding an optimal solution is infeasible. On the other hand, we tackle the problem by using heuristics. In this section, three heuristic algorithms are developed for the problem. The first one is the Greedy heuristic. The second is an improvement of the first one using Local Adjustment heuristic. Thirdly, a global searching method simulated annealing heuristic will be proposed.

A. Greedy Algorithm

Generally speaking, the greedy strategy is often used to solve the optimization problem. We can iteratively select the appropriate tenant assignment that minimizes the objective function. However, in the VMMD problem, there are something important to consider. First, in our constraint, it should ensure that the waiting time at the VM does not exceed $\overline{W}$, and we solve it by assuming the delay of the tenant assignment is Γ when the waiting time at the VM exceed $\overline{W}$, where Γ is a large enough positive number. Then, it can be noticed that each resource, including the VM and the DN, can only be assigned to one tenant. Thus, in our algorithm we need to remove the allocated resources in each iteration.

Algorithm 1 Greedy heuristic

Step 1: Initialization
for all tenant assingment (i, j, k) **do**
  if $\lambda_i - \frac{1}{\alpha_i t_{jk}} + \frac{1}{W} < 0$ **then**
    $d'_{ijk} \leftarrow \lambda_i(d_{ij} + 1/(\frac{1}{\alpha_i t_{jk}} - \lambda_i))$
  else
    $d'_{ijk} \leftarrow \Gamma$
  end if
end for
Step 2: Iterations
while there exists tenant isn't assigned resources **do**
  $d'_{ijk} \leftarrow \min\{d'\}$
  Push (i, j, k, d'_{ijk}) to M
  for all tenant assingment (i', j', k') **do**
    if $i' = i$ OR $j' = j$ OR $k' = k$ **then**
      $d'_{i'j'k'} \leftarrow \infty$
    end if
  end for
end while

The system parameter in Table. I will be initialized in the beginning of Greedy heuristic. According to the method above, the delay matrix d' could be obtained. In each iteration, we find a tenant assignment $< i, j, k >$ so that the delay d'_{ijk} is minimum in the matrix d', and we assign the resources C_j and D_k to A_i. Afterwards, the tenant A_i and the allocated resources C_j and D_k should be removed by setting the element that is related to A_i, C_j and D_k in the matrix d' to infinity. The iteration is terminated when all of the tenants are assigned resources. The detailed algorithm description is shown in Algorithm 1.

B. Local Adjustment Algorithm

Because of the limitation of the service rate of the VM, the Greedy heuristic algorithm may find an infeasible solution, in which there exists an application that the waiting time of its corresponding requests exceeds $\overline{W}$. In order to investigate a better solution, or to possibly find feasible solution in the case

Algorithm 2 Local Adjustment (LA) Heuristic

Step 1: Initialization
The parameter initialization is same to Greedy heuristic
Let M be the solution generated by Greedy heuristic
Step 2: Iteration
while $true$ **do**
  $f \leftarrow false$
  for all z in $\{1, 2, \cdots, l\}$ **do**
    for all z' in $\{i+1, \cdots, l\}$ **do**
      if Find_Better(d',M,z,z',$false$) **then**
        $f \leftarrow true$
        break
      end if
    end for
    if f **then**
      break
    end if
  end for
  if $!f$ **then**
    break
  end if
end while

Algorithm 3 Find_Better (d', M, z, z', SA)

Let (i, j, k, d'_{ijk}) and $(i', j', k', d'_{i'j'k'})$ be M_z and $M_{z'}$
Calculate the adjustable set $C'(i, i')$ and $D'(i, i')$
$s \leftarrow \infty$, $(p_1, p_2) \leftarrow (0, 0)$, $(p'_1, p'_2) \leftarrow (0, 0)$
for all (r, t) in $C'(i, i') \times D'(i, i')$ **do**
  for all (r', t') in $C'(i, i') \times D'(i, i')$ **do**
    if $r \neq r'$ AND $t \neq t'$ AND $d'_{irt} + d'_{i'r't'} < s$ **then**
      $(p_1, p_2) \leftarrow (r, t)$
      $(p'_1, p'_2) \leftarrow (r', t')$
      $s \leftarrow d'_{irt} + d'_{i'r't'}$
    end if
  end for
end for
if $d'_{ijk} + d'_{i'j'k'} > s$ **then**
  $M_z \leftarrow (i, p_1, p_2, d'_{ip_1p_2})$
  $M_{z'} \leftarrow (i', p'_1, p'_2, d'_{i'p'_1p'_2})$
  return $true$
end if
if SA **then**
  //This part is for SA heuristic
  Randomly generate (r, t) and (r', t'), where $r, r' \in C'(i, i')$,$t, t' \in D'(i, i')$,$r \neq r', t \neq t'$
  if $exp((s - d'_{irt} - d'_{i'r't'})/s) > rand(0, 1)$ **then**
    $M_z \leftarrow (i, r, t, d'_{irt})$
    $M_{z'} \leftarrow (i', r', t', d'_{i'r't'})$
  end if
end if
return $false$

of Greedy heuristic fails, we propose Local Adjustment (LA) heuristic that takes the relation between two applications into account.

In order to easily introduce this algorithm, we define one key terminology of our proposed work.

Definition 3: given a solution, assume arbitrary two tenants A_i and $A_{i'}$ are assigned the resource pair $< j, k >$ and $< j', k' >$ respectively. The Adjustable Set of the tenant A_i and $A_{i'}$ is divided into two parts: $C'(i, i')$ and $D'(i, i')$, where $C'(i, i') = \{j, j'\} \bigcup \{j'' | C_{j''} \in C \&\& C_{j''}$ isn't assigned to any tenant in this solution$\}$ is the VM Adjustable Set. $D'(i, i') = \{k, k'\} \bigcup \{k'' | D_{k''} \in D \&\& D_{k''}$ isn't assigned to any tenant in this solution$\}$ is the DN Adjustable Set.

For example, in Fig.1, we assume $l = 3, m = 4, n = 5$, and a solution $\{< 1, 1, 1 >, < 2, 2, 2 >, < 3, 3, 3 >\}$. the adjustable set of A_1 and A_2 is $C'(1, 2) = \{1, 2, 4\}$ and $D'(1, 2) = \{1, 2, 4, 5\}$. Clearly, the tenant A_i and $A_{i'}$ can select resources from $C'(i, i')$ and $D'(i, i')$ without affecting on other tenants.

The LA heuristic starts with a random solution. Since the Greedy algorithm is a convenient way to generate the initial solution, it is applied first. Then, LA algorithm adjusts the assignment with the goal of decreasing the objective function S in each iteration. Note that in order to adjust the assignment, it is necessary to solve the problem that how to find the adjustable tenant assignments. In our algorithm, we consider the relation between two applications. If there exists a tenant pair that makes S decrease by adjusting their assignments without affecting on other tenants, we can reallocate the resources to this two tenants. In order to judge weather there exists the adjustable tenant pair, we traverse all of the tenant pairs. For each tenant pair (i, i'), their adjustable set $C'(i, i')$ and $D'(i, i')$ can be obtained according to Definition 3. Then, we figure out a pair of resource pair that meet the following conditions:

1) They are in the adjustable set $C'(i, i')$ and $D'(i, i')$;
2) They are not the same resource;
3) When they are assigned to A_i and $A_{i'}$ respectively, the sum of their delay denoted by s should be minimum among all the pairs of resource pair that meet the condition 1) and 2).

If s is less than the sum of delay of A_i and $A_{i'}$ in current solution, that is, (i, i') is an adjustable tenant pair, and we reallocate this pair of resource pair to them. Otherwise, we test other tenant pairs. The iteration traverse all of the tenant pairs to find an adjustable tenant pair until there doesn't exist it. Once an adjustable tenant pair is found, it is reallocated, and the algorithm enter the next iteration. The detailed description of LA heuristic is shown in Algorithm 2. The Find_Better sub-procedure used in LA heuristic is described in Algorithm 3, in which the parameter SA is $false$.

C. Simulated Annealing Algorithm

The solutions obtained from Greedy heuristic and LA heuristic are local optimization. Based on LA heuristic, we propose Simulated Annealing(SA) heuristic that is a global

Algorithm 4 Simulated Annealing (SA) Heuristic

Step 1: Initialization
The parameter initialization is same to Greedy heuristic
Let m be the solution generated by LA heuristic
$M \leftarrow m$, $t \leftarrow 0$
Step 2: Iteration
while the number of iterations is $\leq \Omega_1$ **do**
  $f \leftarrow false$
  while the number of tenant pair $\leq \Omega_2$ **do**
    Randomly generate $(A_z, A_{z'})$, where $z \neq z'$
    if Find_Better(d',m,z,z',$false$) **then**
      $f \leftarrow true$
      $t \leftarrow 0$
      break
    end if
  end while
  if !f **then**
    Randomly generate $(A_z, A_{z'})$, where $z \neq z'$
    if !Find_Better(d',m,z,z',$true$) **then**
      $t \leftarrow t + 1$
    else
      $t \leftarrow 0$
    end if
  end if
  if m is better than M **then**
    $M \leftarrow m$
  end if
  if $t \geq \Omega_3$ **then**
    break
  end if
end while

searching algorithm in theory.

It is necessary to find an initial solution for SA heuristic. we can use the solution obtained from LA heuristic. Different from traditional simulated annealing, we use a global parameter M to record the best solution that is found in the second step. We can initialize this parameter by the initial solution. In each iteration, the parameter M is updated to the solution if it's better.

After an initial solution is obtained, the iteration starts. Based on the searching method in LA heuristic, the searching neighborhood $\mathcal{N}$ is defined as all of the tenant pairs. In each iteration, we randomly test Ω_2 tenant pairs from $\mathcal{N}$, where Ω_2 is a positive integer defined in advance. For each tenant pair, we identify it as an adjustable tenant pair by the method from LA heuristic. If it is an adjustable tenant pair, the SA heuristic perform the same way as LA heuristic to reallocate the resources to this pair of tenant, and terminates the test for other tenant pairs. When there doesn't exist the adjustable one in these Ω_2 tenant pairs, we randomly select an tenant pair (z, z') from $\mathcal{N}$, and probabilistically decide whether reallocate resources to them. If (z, z') is an adjustable pair, the probability is 1 to reallocate the pair of resource pair obtained by the way of LA heuristic to them. Otherwise, a pair of resource pair without the same resource from the adjustable set of (z, z') should be generated randomly. For this pair of resource pair, we let s' be the sum of the delay when it is assigned to (z, z'). The probability is $exp(-\Delta/s)$ to reallocate this pair of resource pair to (z, z'), where s is the sum of delay of (z, z') in current solution, and $\Delta = s' - s$. The iteration is terminated until one of the following stopping conditions is met:

1) The number of iterations is greater than Ω_1, where Ω_1 is a predefined positive integer.
2) There doesn't exist the adjustable tenant pair in Ω_3 consecutive iterations, where Ω_3 is a predefined positive integer.

At the end of the iteration, the parameter M is the assignment we need. The detailed description of SA heuristic is shown in Algorithm 4. The Find_Better sub-procedure used in SA heuristic is described in Algorithm 3.

Since the Greedy heuristic is a sub-procedure of the LA heuristic, which is a sub-procedure of the SA heuristic, the result of SA heuristic is always better than that of LA heuristic, and then that of Greedy heuristic, while running time is on the contrary.

V. Numerical Results

This section mainly presents the numerical results to demonstrate the efficiency of Greedy, LA and SA heuristic algorithms. What's more, we conduct comparisons between these algorithms. We first introduce the simulation settings.

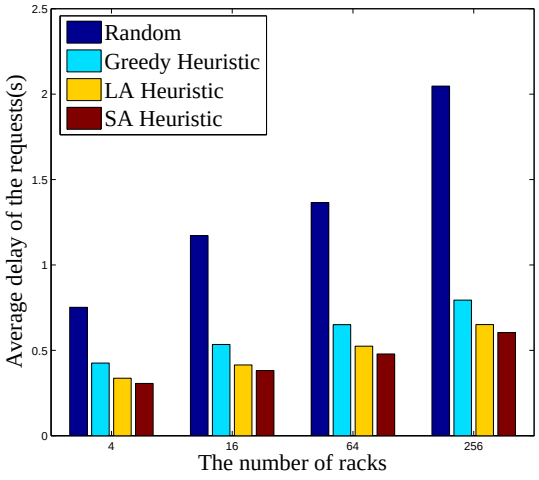

Fig. 2: Average Delay vs. Number of Racks

A. Simulation Settings

We consider a datacenter architecture, which is organized in a hierarchical manner. We assume there are 256 racks in this datacenter, which are used to be assigned to some nodes. The node that belongs to a rack connects to others through

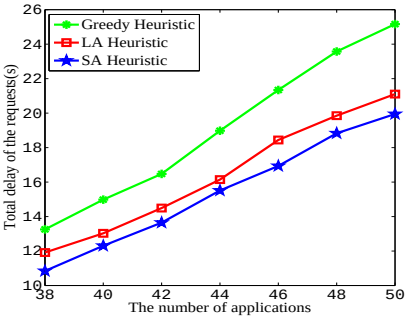

Fig. 3: Total Delay vs. Number of Applications ($K = 16$)

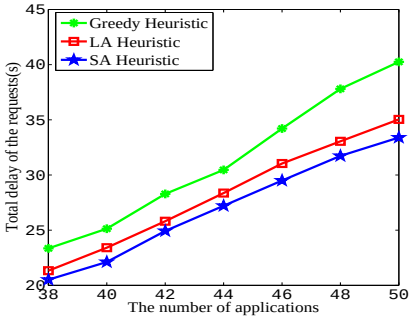

Fig. 4: Total Delay vs. Number of Applications ($K = 256$)

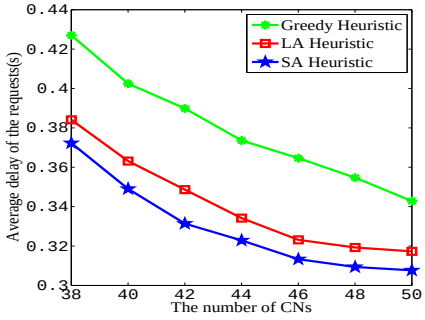

Fig. 5: Average Delay vs. Number of VMs ($K = 16$)

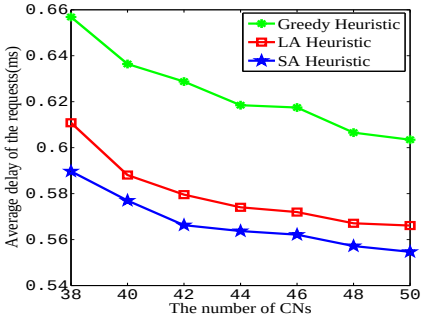

Fig. 6: Average Delay vs. Number of VMs ($K = 256$)

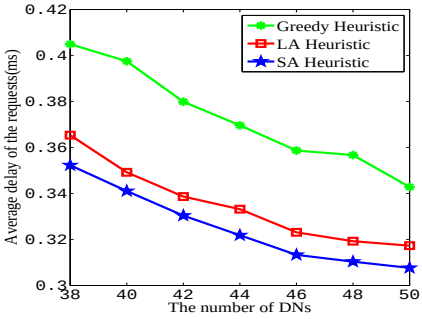

Fig. 7: Average Delay vs. Number of DNs ($K = 16$)

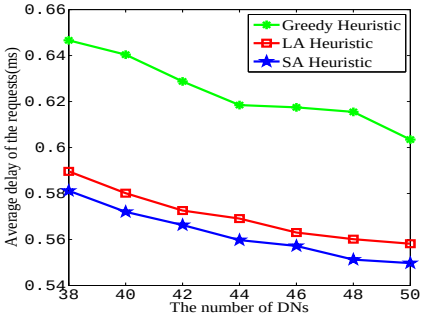

Fig. 8: Average Delay vs. Number of DNs ($K = 256$)

a four-layer structure. In this structure, there is a switch in the first layer, which connects four switches in the second layer, and each of the switches in the second layer connects four switches in the third layer. Similarly, each of the switches in the third layer connects four switches in the fourth layer. Finally, each of the switches in the four layer connects the nodes that belong to racks that are in blocks of 4 (i.e. 0-3,4-7 etc). Clearly, the nodes that connect with the nth layer switch can communication with each other using $(9-2n)$ switches. The required number of VMs and DNs are randomly generated and assigned to one of the racks. While selecting the racks for the VM and DN, we may also restrict it to a subset of racks(e.g. one of the first 16 racks). The latency between a VM and a DN is taken as random between 1 to $k+1$ (ms) per unit data, where k is the number of switches between them.

For the simulations, we assume the cost matrix from tenants to VMs $[d_{ij}]$ are the random value between 5ms-30ms. The request rate $[\lambda_i]$ are randomly chosen from 20-30. The average size of transmission data for each application $[\alpha_i]$ are generated randomly between 1-20 unit data. The expected waiting time upper bound is set as $\overline{W} = 1$s. We examine the performance of our algorithms by 20 times with various link delay values to make our simulation results convincing, and report the average of these examinations.

B. Simulation Results

In order to check the effectiveness of our algorithms. We first observe the performance of our algorithms by comparing them with the randomized approach, in which the resources are assigned to the tenants randomly, under the different number of available racks denoted by K for 4,16,64 and 256. In this part of the simulations, we assume the number of tenants, VMs and DNs are the same, that is $l = m = n = 50$. Fig.2 shows the results on different number of rack sizes. As it can be seen, the average response delay of the requests increases along with the number of racks. This is because as the number of available racks increases, the link delay between VMs and DNs will be increase. Figure also shows that our algorithms can decrease much more average response delay of the requests compared with the randomized approach. The SA heuristic achieves the best performance in our algorithms by decreasing the average response delay for about 68%, while the LA heuristic and Greedy heuristic achieves the second and third optimization with the decreasing percentage as about 64% and 55% respectively.

TABLE II: Success Rate of the Heuristic

The value of K	Greedy Heuristic	LA Heuristic	SA Heuristic
4	100%	100%	100%
16	96%	100%	100%
64	95%	99%	100%
256	90%	95%	99%

The proposed algorithm may fail to find a feasible solution, so the success rate is an important measure to evaluate the algorithms. We calculate the success rate with different number of available racks K. Table II shows the statistic result of the success rate. We can see that the success rate of SA heuristic is higher than LA heuristic, and then in turn Greedy heuristic. The figure also shows that the success rate decreases as number of available racks increases. This is because of the

link delay between VMs and DNs is increasing and the service rate is decreasing as well. In general, all of our algorithms can find a feasible solution with probabilities higher than 90% when the number of available racks is small.

Then, we study the impact of the number of tenants, VMs and DNs under two different number of racks for 16 and 256. The results are shown in Fig.3-8. In each figure, the green curve represents the results of Greedy heuristic, while the red curve represents the results of LA heuristic. The performance of SA heuristic is illustrated by the blue curves. From all of these figures, we can see that the SA heuristic outperforms other approaches in all scenarios, and the performance of LA heuristic is better than that of Greedy heuristic. The delay of SA heuristic decreases about 20% of the Greedy increase on average, while the LA heuristic deceases the delay for about 15% on average. Figures also show that the delay decreases as we reduce the number of racks. The reason is same as the one mentioned in the previous experiment.

The simulation results with the number of tenants l from 38 to 50 and $m = n = 50$ are shown in Fig.3-4. The total response delay is increasing as number of tenants becomes larger. The reason is that when the number of tenants is larger, more resources are needed, which results in the increase of number of waiting queue and more total delay.

The simulation results with the number of VMs from 38 to 50 and $l = 38, n = 50$ are shown in Fig.5-6 and that with the number of DNs from 38 to 50 and $l = 38, m = 50$ are shown in Fig.7-8. We can see that the average response delay of the requests trends down along with the number of VMs (DNs). This is because there are more choices available for tenants, and more DNs (VMs) get closer to the VMs (DNs) as the number of VMs (DNs) increases.

From these simulation results, we can make some conclusions. Our algorithms are efficient in finding a feasible solution. What's more, solutions found by our algorithms result in much less average response delay than the randomized approach. Especially, The SA heuristic achieves the best performance in our algorithms by decreasing the average response delay for about 68%.

VI. Conclusion

In this paper, we study the techniques of VM placement with stochastic requests from the tenants to minimize the total (average) response latency. We model the requests of each application from corresponding tenant as independent Poisson stream. In addition, the VMs are modeled as simple M/M/1 queueing systems, that is, the service time at each VM obeys the exponential distribution. We define VMMD problem and propose three heuristic algorithms: Greedy, Local Adjustment (LA) and Simulated Annealing (SA). According to simulation results, our algorithms are efficient in decreasing the total (average) response latency of the requests from tenants. Especially, the SA heuristic, which decreases the total response latency about 68% on average, shows the best performance on minimizing the total response latency in distributed cloud systems.

Acknowledgment

This paper is supported by the National Science Foundation of China under No. U1301256, 61170058, and 61472383, Special Project on IoT of China NDRC (2012-2766), Research Fund for the Doctoral Program of Higher Education of China No. 20123402110019, the Natural Science Foundation of Anhui Province in China under No. 1408085MKL08.

References

[1] Alicherry, Mansoor, and T. V. Lakshman. "Optimizing data access latencies in cloud systems by intelligent virtual machine placement." *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013.

[2] D. Gross and C.M. Harris, *Fundamentals of Queueing Theory* [M], Wiley: New York, 1998.

[3] Boloor, Keerthana, et al. "Dynamic request allocation and scheduling for context aware applications subject to a percentile response time SLA in a distributed cloud." *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010.

[4] Amazon EC2, "http://aws.amazon.com/ec2".

[5] Cisco Data Center, "http://goo.gl/Sil548".

[6] C. Hyser, B. McKee, R. Gardner, and B. J. Watson. Autonomic virtual machine placement in the data center. Technical report, HP Laboratories, February 2008.

[7] hadoop.apache.org

[8] Li, Xin, et al. "Let's stay together: Towards traffic aware virtual machine placement in data centers." *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014.

[9] Cohen, Reuven, Liane Lewin-Eytan, and Joseph Naor. "Almost optimal virtual machine placement for traffic intense data centers." *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013.

[10] Randles, Martin, et al. "Distributed redundancy and robustness in complex systems." *Journal of Computer and System Sciences* 77.2 (2011): 293-304.

[11] Li, Xin, et al. "Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center." *Mathematical and Computer Modelling* 58.5 (2013): 1222-1235.

[12] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz and E. Silvera. A Stable Network-Aware VM Placement for Cloud Systems. *IEEE International Symposium on Cluster, Cloud and Grid Computing (CCGrid12)*, 2012.

[13] Johnston M, Lee H W, Modiano E. Robust network design for stochastic traffic demands[J]. Lightwave Technology, Journal of, 2013, 31(18): 3104-3116.

[14] Piao, Jing Tai, and Jun Yan. "A network-aware virtual machine placement and migration approach in cloud computing." *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*. IEEE, 2010.

[15] Gao Y, Guan H, Qi Z, et al. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing[J]. Journal of Computer and System Sciences, 2013, 79(8): 1230-1242.

[16] Fang W, Liang X, Li S, et al. VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers[J]. Computer Networks, 2013, 57(1): 179-196.

[17] Dong, Jiankang, et al. "Energy-saving virtual machine placement in cloud data centers." *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013.

[18] Vu, Hieu Trong, and Soonwook Hwang. "A traffic and power-aware algorithm for virtual machine placement in cloud data center." *International Journal of Grid & Distributed Computing* 7.1 (2014): 350-355.

[19] Li, Kangkang, Huanyang Zheng, and Jie Wu. "Migration-based virtual machine placement in cloud systems." *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*. IEEE, 2013.

[20] Huang, Wei, Xin Li, and Zhuzhong Qian. "An energy efficient virtual machine placement algorithm with balanced resource utilization." *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*. IEEE, 2013.

[21] Burkard R E, Cela E. Linear assignment problems and extensions[M]. Springer US, 1999.