A Provident Resource Defragmentation Framework for Mobile Cloud Computing

Weigang Hou, Rui Zhang, Wen Qi, Kejie Lu, Senior Member, IEEE, Jianping Wang, Lei Guo

Abstract—To facilitate mobile cloud computing, a cloud service provider must dynamically create and terminate a large number of virtual machines (VMs), causing fragmented resources that cannot be further utilized. To solve this problem proactively, most existing studies have been based on server consolidation, with the main objective of minimizing the number of active servers. Although this approach can minimize resource fragmentation at a particular time, it may be over aggressive at the price of too frequent VM migration and low system stability. To address this issue, we propose a novel provident resource defragmentation framework that is revenue-oriented with the goal to reduce unnecessary VM migration. Within the proposed framework, we formulate an optimization problem for resource defragmentation at a particular time epoch, with the consideration of the future impact of any VM migration. We then develop an efficient heuristic algorithm that can obtain near-optimal results. Extensive numerical results confirm that our framework can provide the highest profit and can significantly reduce the VM migration cost in practical scenarios.

Index Terms—Cloud computing, Virtual machine (VM), VM migration, defragmentation

1 INTRODUCTION

In recent years, we have witnessed the significant growth of the global mobile economy, with a rapidly increasing number of smartphones and explosively expanding data traffic. According to a recent study by Cisco [1], in 2014, 439 million new smartphones were connected to the mobile network and the total mobile data traffic increased 69 percent worldwide, to 2.5 Exabytes per month by the end of 2014. Such a trend has motivated the emerging of mobile cloud computing [2], [3], with which a smartphone can dynamically offload computationally intensive tasks to remote cloud service providers as so to improve both the performance and the battery life.

Despite the promising future of mobile cloud computing, there are still many challenges to be solved [2], [3], [4], [5], [6], [7]. One of the key challenges the cloud providers face is that, to accommodate the diverse requests from a large number of smartphones, a cloud data center must create and terminate a large number of virtual machines (VM) dynamically. Such frequent VM creation and termination can lead to the resource fragmentation problem, because residual resources (such as CPU, memory, and I/O capacity) may not be fully utilized at a later time [8], [9], [10].

To illustrate such a phenomenon, Fig. 1(a) shows a simple example, in which two servers S1 and S2 are currently hosting three VMs and the resource capacity of each server

is 8 units. Furthermore, VM1 and VM3 occupy 2 units of resources, and VM2 occupies 3 units of resources. In such an example, if a new VM requires 7 units of resources, it cannot be accommodated with the current VM placement. In this paper, we refer to this problem as the fragmentation problem because it occurs due to fragmented resources in different servers.

Existing approaches dealing with fragmentation in cloud can be classified into two categories: (1) VM migration upon new VM request arrivals and (2) proactive server consolidation. In the first category, when a new VM request arrives, the cloud data center will determine whether it can accommodate the request with existing available resources [11], [12], [13]. If such a placement is not possible, the data center may migrate one or more existing VMs and then place the new VM into the system [14]. Although this approach is viable, when there are hundreds of VMs that need to be provisioned in a short time, many VM migrations may be invoked, which will certainly consume extra bandwidth and cause service degradation to existing VMs. In the literature, some recent studies have discovered that VM migrations can lead to significant migration cost and delay [15], [16].

In the second category, server consolidation can be performed before the arrival of new VM requests. In recent
years, server consolidation has been investigated extensively in the literature [11], [13], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]. Usually, an outcome of server consolidation is that existing VMs are packed into the minimal number of servers, which can reduce resource fragments. Another outcome is that some servers that do not host any VM will be shut down to conserve energy. In Fig. 1(b), with server consolidation, VM3 will be migrated to server $S_1$, and thus $S_2$ can accommodate the new VM request that requires 7 units of resources.

Despite the importance of server consolidation, we observe that there are two major issues that have not yet been addressed. First, shutting down a server may not be desirable if the same server shall be restarted after a short period due to the increasing demands. Second, also more important, server consolidation may lead to excessive VM migrations because some residual resources may be utilized in the near future. For instance, in Fig. 1(a), if there are two new VMs to arrive, requiring 5 and 3 units of resources, respectively, they can be directly assigned without any VM migration. In other words, migrating VM3 is unnecessary in such a case.

Motivated by the observation above, in this paper, we propose a provident resource defragmentation framework, where the main idea is to migrate VMs as few as necessary, based on an estimation of future demands. In our framework, a major challenge is how to make two inter-correlated decisions, namely, when to conduct defragmentation and which VMs need to be re-provisioned in one defragmentation.

To address this challenge, we formulate an optimization problem to avoid unnecessary VM migrations, given a set of existing VMs and an expected set of incoming and terminating VMs. After showing that the problem is NP-hard, we develop an efficient heuristic algorithm that can obtain a near-optimal solution. To the best of the authors' knowledge, this is the first study that systematically investigates the defragmentation problem in a cloud data center. Within this scope, our main contributions are summarized as follows.

- We propose a general framework for resource defragmentation with the objective to avoid unnecessary VM migrations. In our framework, defragmentation can be fine-tuned by determining when to trigger the defragmentation operation and how to migrate existing VMs. Our framework is general in that it can also be applied to other operations, such as VM placement and server consolidation.
- Within the proposed framework, we formulate an optimal resource defragmentation problem for an arbitrary time epoch. To avoid unnecessary VM migrations, we consider that each VM is associated with an accommodation revenue and a migration cost. We then formulate the problem to maximize the profit (i.e., total revenue minus costs) with a prediction of future demands.
- In our formulation and algorithm design, we consider a demand prediction that requires only the estimation of the total number of arrival and the request distribution of different VM instances. Our numerical results show that such an approach can lead to reasonably good performance even if we do not have an accurate estimation of the number of arriving VMs.

The rest of the paper is organized as the following. First, we discuss related work in Section 2. We then elaborate on the framework for provident resource defragmentation in Section 3. Based on the proposed framework, we formulate the optimal provident resource defragmentation problem in Section 4. Since the problem is hard by nature, we develop an efficient heuristic algorithm in Section 5. Finally, we present numerical results in Section 6, before concluding the paper.

2 Related Work

In this section, we review the related work on VM placement, VM migration and server consolidation. The problem of VM placement is essentially multi-dimensional vector packing [30]. Given a set of VM requests and vacant cloud resources, VM placement aims to maximize the overall utility. Some VM provisioning also considers live VM migration [14].

Server consolidation has been widely considered as an approach to optimize various utility measurements, such as balancing server workload [21], [22], [23], maintaining the reliability of hardware devices [24], enhancing scalability of server consolidation algorithm [25], preserving fairness [26] and reducing operational cost [31]. Although live VM migration gives the cloud vendors more freedom on optimizing their utility measurements, it also induces extra operational cost and may disrupt the service of existing VMs. Reducing such migration cost is the major focus of this paper.

Leveraging on the knowledge of reliable prediction on future VM requests, unnecessary VM migrations may be avoided. He et al. [12] utilized a multi-dimensional normal distribution of historical VM requests for predicting the resource requirement of future VM requests. When the residual resources on a server are abundant to satisfy the mean resource requirement of a VM request, this server will not be considered for server consolidation. However, in reality, a cloud service provider may offer several types of VM instances. Therefore, using the mean resource requirement of VM requests may still lead to unnecessary VM migrations. In the literature, there are several other models for predicting the changes of the future resource demands of VMs [22]. In this study, we consider a much weaker assumption in that, for each type of VMs, we only use the expected number of arrivals at a future time epoch.

3 A Provident Resource Defragmentation Framework

In this section, we propose a provident resource defragmentation framework to address the resource fragmentation issue. We first briefly explain the main idea of the proposed framework. We then elaborate on important components of the framework.
Fig. 2. A defragmentation framework.

3.1 The Main Idea

As introduced previously, the main idea of our framework is to perform VM migrations as few as necessary. Here “necessary” first means that the defragmentation operation shall aim to improve the resource utilization of a data center. Moreover, it also means that some fragments can be kept without defragmentation if they can be used to accommodate expected future VM requests. For instance, in Fig. 1(a), we do not need to migrate VM3 if we predict that an incoming VM will require up to 6 units of resources.

Besides the discussions above, we also consider that defragmentation shall be performed over time as a process, because the status of the data center is varying due to the arrival and termination of VMs.

3.2 The Framework

To address the fragmentation problem, we propose a framework that includes the following key components, which are also shown in 2.

- Demand estimation: This component observes and records the demands of VM arrival and departure. Based on the historical data and instantaneous demands, it can estimate and predict future VM arrival events. In this module, various estimation schemes and algorithms can be adopted. The output of this module will be the arrival and departure models that will be used by the fragmentation monitoring module.

- Fragmentation monitoring: This component keeps monitoring the fragmentation condition of a data center and determines when the defragmentation operation shall be performed. Such a signal will be forwarded to the defragmentation module if defragmentation is necessary.

- Defragmentation: This component determines how to minimize the VM migration cost while obtaining the best possible system revenue by accommodating expected VMs. One output of this module is the schedule to migrate some existing VMs in the data center. If some VMs have been migrated, the updated VM allocation will be sent to the VM placement module.

- VM placement: This component terminates VMs on-demand and also determines how to migration existing VMs and how to place arrived VMs into the data center so as to maximize the overall profit of the data center.

4 An Optimal Provident Resource Defragmentation Problem

In this section, we first introduce the system model and key notations. For an arbitrary time epoch, we then formulate the problem and discuss its complexity.

4.1 The System Model

In our study, we consider a general cloud data center that consists of \( S \) networked physical servers. For each server \( j \) (\( 1 \leq j \leq S \)), we let \( O_j \) be the fixed capacity; we also consider that the resource utilization is time-varying, denoted as \( L_j \) (\( L_j \leq O_j \)); we finally define \( F_j = O_j - L_j \), which means the remaining resources in server \( j \). We shall note that, for a more general case that a server has various types of resources (e.g., disk I/O, CPU, and memory), we merely need to replace the current real variables with multidimensional vectors, namely, \( \tilde{O}_j, \tilde{L}_j \) and \( \tilde{F}_j \).

Following the common practice (such as Amazon EC2), we assume that the cloud data center accepts \( C \) types of different VM requests. Each VM request can be represented by a 3-tuple: \( < c, e_c, r_c > \), where \( c \) is the type index, \( e_c \) is the payment, and \( r_c \) is the resource requirement. We also consider that \( r_c \) is a number and \( r_1 > r_2 > \cdots > r_C \), and 

\[
\begin{align*}
e_1 > e_2 > \cdots > e_C, \text{ i.e., a request consuming more resources will generate a higher revenue, a very reasonable assumption. Similarly, when we consider the diverse resource requirements, } r_c \text{ will be changed into } r_{ce}. 
\end{align*}
\]

At a particular time epoch \( t \), there are some existing VMs in the cloud data center, there can also be some newly arrived VMs. In our problem, we also assume that an estimated VM arrival for a future time epoch \( t' \) is available. In other words, we consider a two-period problem, i.e., a current time \( t \) and a future time \( t' \), where some VM assignment and migration decisions have to be made at time \( t \) with the consideration of the future impact at time \( t' \).

4.2 Notation Definitions

To facilitate further discussions, we summarize important notations below with an alphabetic order.

- \( A \) (with subscription and superscription): A two dimensional VM assignment matrix.
- \( C \): The total number of VM types.
- \( e_c \): The revenue of accommodating a type-c VM.
- \( F_j \): The remaining resources in the \( j^{th} \) server.
- \( G_c \): The set of servers that contain VMs to be migrated.
- \( G_i \): The set of servers that have available resources.
- \( i \): The index of a VM.
- \( j \): The index of a server.
- \( \ell_c \): The migration cost coefficient (\( 0 < \ell_c < 1 \)).
- \( \ell_u \): The server cost coefficient (\( 0 < \ell_u < 1 \)).
- \( L_j \): The resource utilization of the \( j^{th} \) server.
- \( m_c \): The number of existing type-c VMs in the system.
- \( n_c \): The number of type-c VMs arrived at \( t \).
4.3 The Problem Formulation

With the system model and assumptions, we can now investigate the optimal defragmentation problem at a particular time \( t \). Next, we first consider a basic VM placement problem with migration. We then discuss how to formulate the defragmentation issue.

4.3.1 The Optimal VM Placement with Migration Problem

At time \( t \), let an \( m_c \times S \) matrix \( A^{t,c} \) record the assignment of all existing type-\( c \) VMs. Specifically, the element of this matrix, denoted as \( a(c)_{ij} \), is 1 if the \( i \)-th type-\( c \) VM is placed in the \( j \)-th server; otherwise \( a(c)_{ij} = 0 \). And we have

\[
\sum_{j=1}^{S} a(c)_{ij} = 1, \forall i \in [1, m_c], \forall c \in [1, C],
\]

and

\[
\sum_{c=1}^{C} r_c \cdot \sum_{i=1}^{m_c} a(c)_{ij} \leq O_j, \forall j \in [1, S].
\]

To accommodate newly arrived VMs, some existing VMs may be migrated. To formulate this case, we let an \( m_c \times S \) matrix \( A^{t,c'}(c) \), with elements \( a(c')_{ij} \), record the assignment of all existing type-\( c' \) VMs after VM migrations, where the definition of \( a(c')_{ij} \) is similar to that of \( a(c)_{ij} \). With such a definition, we have

\[
a(c)_{ij} \cdot [1 - a(c)_{ij}] = 1,
\]

if and only if the \( i \)-th type-\( c \) VM has been migrated from server \( j \).

We further consider that, at time \( t \), there are \( n_c \) type-\( c \) VM arrivals. We let an \( n_c \times S \) matrix \( A^{t,\alpha}(c) \) record the assignment of the newly arrived type-\( c \) VMs. We let the element of \( A^{t,\alpha}(c) \) be \( \alpha(c)_{ij} \), where the definition of \( \alpha(c)_{ij} \) is similar to that of \( a(c)_{ij} \).

Consequently, we can calculate the total revenue by

\[
\pi = \sum_{c=1}^{C} e_c \cdot \sum_{i=1}^{n_c} \sum_{j=1}^{S} \beta(c)_{ij}.
\]

Next, the total migration and server costs can then be represented by

\[
k = \sum_{c=1}^{C} \sum_{i=1}^{m_c} \sum_{j=1}^{S} \{ \ell_r \cdot r_c \cdot a(c)_{ij} \cdot [1 - a(c)_{ij}] \} + \sum_{j=1}^{S} \ell_u \cdot \omega_j,
\]

where \( \forall j \in [1, S] \),

\[
\omega_j = \begin{cases} 
0, & \text{if } \sum_{i=1}^{m_c} a(c)_{ij} + \sum_{i=1}^{n_c} \beta(c)_{ij} = 0; \\
1, & \text{otherwise}.
\end{cases}
\]

To formulate the problem, the above matrices shall satisfy a number of constraints.

\[
\sum_{j=1}^{S} a(c)_{ij} = 1, \forall i \in [1, m_c], \forall c \in [1, C],
\]

and

\[
\sum_{j=1}^{S} \beta(c)_{ij} \leq 1, \forall i \in [1, n_c], \forall c \in [1, C],
\]

and

\[
\sum_{c=1}^{C} r_c \cdot \left[ \sum_{i=1}^{m_c} a(c)_{ij} + \sum_{i=1}^{n_c} \beta(c)_{ij} \right] \leq O_j, \forall j \in [1, S].
\]

Here, Eq. (7) ensures that no VM can be divided. Eq. (8) indicates that some VM requests could be blocked even with VM migrations. Eq. (9) indicates that the number of accommodated VMs is constrained by the server capacity.

With the definitions and constraints above, we can formulate the problem of VM placement and migration by the following objective function:

\[
\text{Maximize } (\pi - \kappa).
\]

Here, we note that the migration cost can be skipped if VM migrations are not allowed, which means that the problem will be degenerated to a problem that is similar to the classic bin-packing problem.

4.3.2 The Optimal Provident Resource Defragmentation Problem

In our framework, we can predict the demands to arrive at a future time epoch \( t' \) with a much weak assumption that we merely need to estimate the total number of VM arrivals and the proportion of each type of VM instances. Based on these information, provident resource defragmentation can be executed so as to improve the utilization and to reduce the waiting time before assigning expected VMs at \( t' \). Nevertheless, with provident resource defragmentation, some existing VMs could be migrated, which leads to a higher migration cost that shall be taken into account.

To formulate the optimal provident resource defragmentation problem, we first assume that by virtue of some simple predictions, we can obtain \( p_c(n'_c) \) which is the probability that there are \( n'_c \) expected type-\( c \) VMs to arrive at \( t' \). We take this probability to reflect the prediction accuracy. For example, when \( p_c(n'_c) = 1 \), it means that the prediction accuracy is up to 100%. Then we can define an \( n'_c \times S \) matrix \( A^{t,\beta'}(c) \) that records the assignment of expected type-\( c \) VMs. Similar to previous definitions, we let \( \beta'(c)_{ij} \) represent the proportion of type-\( c \) VMs in the \( j \)-th server at time \( t' \).
be the element of $A_{t', \beta'}(c)$; and we let $\beta'(c)_{ij} = 1$ if the $i^{th}$ expected type-$c$ VM is placed in the $j^{th}$ server, otherwise, $\beta'(c)_{ij} = 0$.

With this definition, we can derive the estimated revenue by

$$\pi' = \sum_{c=1}^{C} e_c \sum_{n'_{c}=0}^{\infty} \left\{p_c(n'_{c}) \cdot \sum_{i=1}^{S} \sum_{j=1}^{\infty} [\beta'(c)_{ij}] \right\}. \quad (11)$$

Since the cost is associated only with existing VMs, it can still be represented by Eq. (5).

Similar to the previous problem, the placement of VMs shall satisfy Eq. (7), and the following constraints:

$$\sum_{j=1}^{C} \beta'(c)_{ij} \leq 1, \forall i \in [1, n'_{c}], \forall c \in [1, C], \quad (12)$$

and

$$\sum_{c=1}^{C} r_c \left[ \sum_{i=1}^{S} \alpha(c)_{ij} + \sum_{i=1}^{\infty} \beta'(c)_{ij} \right] \leq O_j, \forall j \in [1, S]. \quad (13)$$

Clearly, an optimization problem can be formulated with the objective:

Maximize $$(\pi' - \kappa). \quad (14)$$

Note that Eq. (11), as a generic formulation, includes an infinite number of states. In practice, the number of VM requests at any time is bounded. So to study a practical version of the problem, with the probability $\prod_{c=1}^{C} p_c(n'_{c})$, we can obtain an estimated total number of VMs to arrive at $t'$, denoted as $n'$. Similarly, when $\prod_{c=1}^{C} p_c(n'_{c}) = 1$, the prediction accuracy is 100%. We then obtain a vector $\Phi = \{\phi_c\}_{c \in [1, C]}$, where $\phi_c$ is the proportion of type-$c$ VM requests over all incoming VM requests at $t'$ and we have $\sum_{c=1}^{C} \phi_c = 1$.

In this manner, $n'_{c}$ can be simply estimated by

$$n'_{c} = n' \times \phi_c. \quad (15)$$

Consequently, we can keep all the constraints in Eq. (7), Eq. (12), and Eq. (13); and we can modify the revenue by

$$\tilde{\pi}' = \sum_{c=1}^{C} e_c \sum_{n'_{c}=0}^{n'_{c}'} \left\{p_c(n'_{c}) \cdot \sum_{i=1}^{S} \sum_{j=1}^{\infty} [\beta'(c)_{ij}] \right\}. \quad (16)$$

Finally, based on the estimations of the total number of VM arrivals and the proportion, the optimal provident resource defragmentation problem can be formulated with the following objective function:

Maximize $$(\tilde{\pi}' - \kappa). \quad (17)$$

and we can analyze the complexity of this problem in the following theorem.

**Theorem 1.** The optimal provident resource defragmentation problem is NP-hard.

**Proof.** For the problem of VM placement and migration formulated in Eq. (10), if we skip the migration cost, this problem is degraded into the classic NP-hard bin-packing problem with the objective function that has $C \cdot n_c \cdot S$ variables (see Eq. (4)), and the constraints totally have $(S + C \cdot m_c)$ variables (see Eq. (1) and Eq. (2)). Then, the storage complexity of this classic NP-hard problem is approximately $(C \cdot n_c \cdot S) + (S + C \cdot m_c)$. For example, if we assume that $C = 10$, $n_c = m_c = 10^3$ and $S = 10^2$, then this problem will have 1, 010, 100 variables.

For our optimal provident resource defragmentation problem with the objective function (see Eq. (17)) that has $C \cdot (n'_{c} + m_c) \cdot S$ variables, the constraints (Eq. (7), Eq. (12), and Eq. (13)) totally have $[S + S \cdot (C \cdot m_c + n'_{c})]$ variables. As a result, the optimal provident resource defragmentation problem will have 2, 020, 200 variables if $n'_{c}$ is $10^3$. Therefore, our problem is also NP-hard because it has more variables during the process of solving the optimal solution. □

**5 AN EFFICIENT ALGORITHM FOR PROVIDENT RESOURCE DEFRAGMENTATION**

In the previous section, we have formulated an optimal provident resource defragmentation problem. Since the problem is NP-hard, in this section, we develop an efficient algorithm to solve it. Since we are focused on the migration cost, we ignore the server cost in the algorithm. In the rest of this section, we first present the main procedure of the algorithm. We then elaborate on the details of the algorithm.

**5.1 The Provident Resource Defragmentation (PRD) Algorithm**

In our PRD algorithm, which is illustrated in Algorithm 1, we consider that the following parameters are...
given:
- $A_\ell^\alpha$: The allocation matrix for the current status of the cloud data center. Specifically, $A_\ell^\alpha$ consists of all $A_\ell^\alpha(c)$ we defined in the last section.
- $\Phi$: The vector that consists of the proportion of each type of VM.
- $n^t$: The expected number of VMs to arrive at $t^t$.

With the above inputs, we first execute a biggest volume first (BVF) algorithm, in which we place expected VMs into servers one-by-one, from the VM with the largest volume. The output of the BVF is the tentative allocation matrix for expected VMs, denoted as $A_\ell^{\beta'}(0,0)$. We can also obtain a vector $n^t = \{n^t_c\}_{c \in [1,C]}$, where element $n^t_c$ represents the number of expected type-$c$ VMs that can be assigned. Then we can obtain the revenue and cost after this step as the following:

$$\tilde{\pi}_{0,0} = \sum_{c=1}^{C} c_e \cdot n^t_c,$$

and $n_{0,0} = 0$.

If all expected VMs can be allocated, we stop the algorithm. Otherwise, starting from the smallest index $c$ where $n^t_c < n^t_0 = \phi_c$, we execute an iterative algorithm, namely, maximum profit defragmentation (MaxPD). Each time we execute the MaxPD algorithm, we obtain the maximum profit by migrating existing VMs and allocating expected VMs, based on the best allocation matrices we obtained in the previous step. In particular, we let $A_\ell^{\alpha'}(c,k_c)$ and $A_\ell^{\beta'}(c,k_c)$ be the VM assignment for existing and expected VMs if $k_c(0 \leq k_c \leq n^t_c - n^t_0)$ expected type-$c$ VMs can be accommodated after the BVF assignment. For each pair of $A_\ell^{\alpha'}(c,k_c)$ and $A_\ell^{\beta'}(c,k_c)$, we can determine the revenue and cost, denoted as $\tilde{\pi}_{c,k_c}$ and $\kappa_{c,k_c}$, respectively. We further define $k^*_c$ to be the $k_c$ that leads to the maximum profit $v_{c,k_c} = \tilde{\pi}_{c,k_c} - \kappa_{c,k_c}$.

Note that we can calculate $\tilde{\pi}_{c,k_c}$ recursively with:

$$\tilde{\pi}_{c,k_c} = \tilde{\pi}_{c-1,k_c-1} + k_c \cdot c_e,$$

where $k^*_0 = 0$. On the other hand, $\kappa_{c,k_c}$ can be obtained by comparing $A_\ell^{\alpha'}(c,k_c)$ and $A_\ell^{\gamma'}(c,k_c)$ using

$$\kappa_{c,k_c} = \sum_{i=1}^{S} \sum_{j=1}^{\alpha(c,k_c)} \{\ell_r \cdot r_c \cdot (a(c,k_c))_{ij}[1 - \alpha(c,k_c)]_{ij}\},$$

where we let $\alpha(c,k_c)_ij$ be the element of $A_\ell^{\alpha'}(c,k_c)$.

Finally, after executing the PRD algorithm, we can determine the best allocation matrices $A_\ell^{\alpha'}$ and $A_\ell^{\beta'}$. Here, $A_\ell^{\alpha'}$ consists of all $A_\ell^{\alpha'}(c,k_c)$ we define above. The definition of $A_\ell^{\beta'}$ is similar to that of $A_\ell^{\alpha'}$.

Fig. 3 illustrates the main idea of the PRD algorithm. We can observe that, for each type of VM, the revenue increases linearly with the increase of $k_c$. However, due to the migration cost, the overall profit may not be increased for some $k_c$. Therefore, we design the MaxPD algorithm to find the best solution $k^*_c$ that leads to the maximal profit. In each iteration of MaxPD, we try to accommodate one expected type-$c$ VM, by using a minimum cost migration (MinCM) algorithm, until all $n^t_c - n^t_0$ expected VMs are placed or the residual resources are not sufficient to support more type-$c$ VMs.

### 5.2 The Biggest Volume First (BVF) Algorithm

The BVF algorithm is shown in Algorithm 2. In this algorithm, start from the VM with the largest volume, we go through all types of expected VMs in an increasing order. Specifically, for a particular type $c$, we traverse all possible servers and determine whether we can accommodate some expected type-$c$ VMs in each server, until all servers have been evaluated or all VMs have been assigned.

To illustrate the operation of the BVF algorithm, we show a simple example in Fig. 4(a) and (b), where $C=3$, $r_1=16$, $c=0.2$, $r_2=8$, $\phi_2=0.5$, $r_3=2$, $\phi_3=0.3$ and $O_j=40, \forall j \in [1,8]$. Assuming the status, $A_\ell^{\alpha'}$, of the cloud is shown in Fig. 4(a). We assume that $n^t=10$. With the BVF algorithm, we order to determine $n^t_1$, $n^t_2$ and $n^t_3$. Fig. 4(b) demonstrates that $n^t_1 = 0$ due to fragmented resources. Next, we try to place expected type-2 VMs and we notice that all of them can be accommodated. That is $n^t_2=n^t_0 \cdot \phi_2 = 5$. Finally, we try to assign expected type-3 VMs and all of them can be assigned as well, i.e., $n^t_3 = n^t_0 \cdot \phi_3 = 3$. Therefore, we have $n^t_1 = 0 < n^t_0 \cdot \phi_1$, $n^t_2 = n^t_0 \cdot \phi_2$, and $n^t_3 = n^t_0 \cdot \phi_3$.

### 5.3 The Maximum Profit Defragmentation (MaxPD) Algorithm

As explained previously, in this algorithm, we try to find the maximum profit when assigning expected type-$c$ VMs. The details of the algorithm can be found in Algorithm 3.

It shall be noted that, during the VM assignment process, some existing VMs may be migrated. Therefore, it is important to avoid the ping-pong effect, which means that one VM will be migrated back and forth. In our design, we avoid such ping-pong effect by enforcing a one-way migration policy that prevents any VM to be migrated back to its original server.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2477778, IEEE Transactions on Emerging Topics in Computing

In the example of Fig. 4(b), we can observe that $c^* = 1$, and we can also see that the remaining resources of every server is less than $r_{c^*} = 16$. So only type-2 and type-3 VMs can be migrated. In addition, we have $G_s = \{1, 2, 3, 4, 5, 6\}$ and $G_t = \{2, 3, 6, 7, 8\}$.

5.4 The Minimum Cost Migration (MinCM) Algorithm

The purpose of the MinCM algorithm is to assign one expected type-$c$ VM with the smallest migration cost. The main idea is to identify a server in $G_s$ such that we can make space for this expected type-$c$ VM by migrating VMs from this server to servers in $G_t$. The details of the algorithm can be found in Algorithm 4.

As shown in this algorithm, we evaluate all possible servers in $G_s$ one-by-one, according to the remaining resources, from the largest to the smallest. Particularly, in each round, we find the server $j^*$ in $G_s$ that has the largest vacant space. Next, if $j^*$ is in $G_t$, we define $G'_t = G_t - \{j^*\}$.

To determine whether we can accommodate the expected type-$c$ VM in server $j^*$, we first count the number of existing VMs that can be migrated. In other words, we count the number of existing type-$c'$ VMs in server $j^*$, denoted as $f_{c'}(j^*)$, $c^* = c' \leq C$.
Next, we consider whether there are enough remaining resources in \( G'_t \). Suppose we have a sufficiently large number of type-\( c' \) VMs (\( c' < c' \leq C \)), we can use the BVF algorithm to determine the maximum number of type-\( c' \) VMs that can be assigned, denoted as \( f_{c'}(G'_t) \), \( \forall c' < c' \leq C \).

With \( f_{c'}(G_t) \) and \( f_{c'}(j^*) \), we can determine the maximum number of type-\( c' \) VMs that can be moved out of server \( j^* \), denoted as \( f_{c'} \), where \( f_{c'} = \min \{ f_{c'}(j^*), f_{c'}(G'_t) \} \).

We can then judge whether we can accommodate the expected type-\( c \) VM by checking
\[
\left\lfloor \sum_{c'=c+1}^{C} (f_{c'} \cdot r_{c'}) + F_{j^*} \right\rfloor \geq r_c. \tag{24}
\]

If the inequality above holds, we execute an increasing first (IFF) algorithm shown in Algorithm 5 to migrate existing VMs to servers in \( G'_t \). Otherwise, we let \( G'_s = G'_s - \{ j^* \} \) and continue the while loop until \( G'_s \) is empty.

As mentioned above, server \( j^* \) will migrate out some type-\( c' \) VMs (\( \forall c' < c' \leq C \)) so that it has enough space of carrying type-\( c \) VMs. After that, if this server \( j^* \) will become the target server to receive the same group of type-\( c' \) VMs it previously migrated out, the ping-pong effect will occur in the PRD algorithm. Therefore, to avoid this ping-pong effect, we add a condition such that server \( j^* \) no longer functions as the target server in the following process, i.e., the residual resource becomes smaller than \( r_c \) after the server \( j^* \) receives type-\( c \) VMs. Note that, the qualification of being a target server can be seen in Eq. (22). Above all, the condition of avoiding ping-pong effect can be represented in the following equation.
\[
\left\lfloor \sum_{c'=c+1}^{C} (f_{c'} \cdot r_{c'}) + F_{j^*} \right\rfloor \% r_c < r_c. \tag{25}
\]

We now use the example in Fig. 4(b) and (c) to illustrate the operation of the MinCM algorithm. In Fig. 4(b), we can identify \( j^* = 6 \). According to our definition, \( f_2(6) = 2 \) and \( f_0(6) = 0 \). On the other hand, since server 6 is in \( G_t \), we temporarily remove it so that \( G'_t = \{ 2, 3, 7, 8 \} \). Consequently, we can calculate \( f_2(G'_t) = 2 \) and \( f_0(G'_t) = 4 \). Clearly, \( f_2 = 2 \) and \( f_0 = 0 \). Since \( F_0 + f_2 \cdot r_2 = 8 + 2 \times 8 = 24 > r_1 = 16 \), we can see that an expected type-1 VM can be accommodated. By using the IFF algorithm, we can migrate one type-2 VM from server 6 to server 7 and then place an expected type-1 VM into server 6, as shown in Fig. 4(b). In a similar way, we can accommodate another expected type-1 VM by identifying \( j^* = 2 \), as well as by migrating a type-2 VM from server 2 to server 8 and migrating a type-3 VM from server 2 to server 3, as illustrated in Fig. 4(c).

With the algorithms above, we can prove the following theorem.

**Theorem 2.** Ping-pong effect does not exist in the PRD algorithm.

**Proof.** As in Eq. (25), we have added the condition of avoiding ping-pong effect into MinCM shown in Algorithm 4. This condition is not very rigid because: (1) \( \forall j^* \), \( F_j \) does not decrease after executing the MinCM algorithm, (2)
from \{40, 60, 80\}. To simulate dynamic demands, we consider a certain duration that has been partitioned into equal-sized time slots [14]. We assume that VMs can arrive only at the beginning of a time slot. We also assume that each VM can last for a number of slots, which is a random number chosen uniformly in \([1, 4]\). Moreover, we consider that there are \(C = 3\) types of VMs, with \(r_1 = 16, r_2 = 8\) and \(r_3 = 2\), as illustrated in Fig. 4. We also let \(c_v = r_v\). As to the proportion of VMs, we assume that \(\Phi = (0.2, 0.5, 0.3)\).

In our experiments, we evaluate mainly four performance metrics: (1) the total profit in 20 slots, (2) the average migration cost per operation, (3) the average service delay per operation, and (4) the running time. We present results particularly, about comparison, our PRD algorithm can achieve near-optimal optimality, we also present an upper bound for each case could be an important parameter to periodic consolidation because there exists an optimal \(T\) that leads to the best profit. Nevertheless, we can observe from the table that the proposed PRD algorithm outperforms all existing approaches with different \(\pi\) settings. To demonstrate its optimality, we also present an upper bound for each case in Table 1, which is obtained by assuming that all servers are replaced by one single server with aggregated resources so that there is no fragmentation. As we can see from the comparison, our PRD algorithm can achieve near-optimal results, particularly, about 6\% average convergent rate, in different \(\pi\) cases.

To show the robustness of our method, in Table 2, we compare different approaches in Scenario 2, where we choose \(\ell_r = 0.01\) but let \(n' = 230\), which means that our estimation of \(n'\) is not totally accurate. We can observe from the results that the total profit increases with the increase of \(S\), which is reasonable because a larger \(S\) means more resources to accommodate VM requests. The numerical results also show that, even with an inaccurate estimation of \(n'\), our algorithm can still lead to the best profit performance that is close to the upper bound.

![Fig. 5. The average migration cost in Scenario 1 and Scenario 2.](image1)

![Fig. 6. The average service delay in Scenario 1 and Scenario 2.](image2)

### 6.3 The Average Migration Cost and Service Delay

To understand the impact of migration overheads, we define average migration cost as the total migration cost during the designated duration divided by the total number of times that the defragmentation (or consolidation) operations are performed. For example, our PRD algorithm is performed in each time slot. Therefore, the average migration cost is the total cost in 20 slots divided by 20. On the other hand, for periodic server consolidation with \(T = 5\), the average migration cost is the total cost in 20 slots divided by \(3 = \frac{20}{5} - 1\).

The definition of the average service delay is similar to the aforementioned definition for migration costs. Here we note that, in practice, the total migration delay depends on many factors, as shown in [32]. In this study, to simplify the discussions, we consider that the service delay is 40 seconds during the precopy-based migration of 512MBytes memory, which is an experimental result evaluated in the data center network with GB-level bandwidth. In our experiments, we assume that 512MBytes memory is one unit of resource, so the service delay of migrating three types of VMs are 80, 320, and 1280 seconds, respectively. Moreover, we assume that VMs shall be migrated consecutively.

In Fig. 5(a) and (b), we compare the average migration cost of our scheme with periodic consolidation, for Scenarios 1 and 2, respectively. Since the average costs are similar for periodic consolidation with different \(T\), we only show the results for \(T = 1\) and \(T = 5\) for demonstration. We can observe that, the proposed algorithm can significantly reduce the average migration cost (e.g., less than 2\% of that of periodic consolidation), which meets our design objective. These results suggest that, with the proposed PRD algorithm, the cloud data center can accommodate more VM requests and have much fewer VM migrations to achieve this goal.

Another interesting observation from Fig. 5 is that, in
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2015.2477778, IEEE Transactions on Emerging Topics in Computing

TABLE 1
The total profit in Scenario 1.

<table>
<thead>
<tr>
<th>VMs</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
<th>350</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure placement</td>
<td>16434</td>
<td>16816</td>
<td>16806</td>
<td>17036</td>
<td>16858</td>
</tr>
<tr>
<td>T=1</td>
<td>16473.26</td>
<td>16818.5</td>
<td>16797.58</td>
<td>17012.86</td>
<td>16870.78</td>
</tr>
<tr>
<td>T=2</td>
<td>16495.44</td>
<td>16821.10</td>
<td>16761.12</td>
<td>17033.34</td>
<td>16865.18</td>
</tr>
<tr>
<td>T=3</td>
<td>16410.96</td>
<td>16795.5</td>
<td>16682.92</td>
<td>16976.12</td>
<td>16796.08</td>
</tr>
<tr>
<td>T=4</td>
<td>16504.72</td>
<td>16822.58</td>
<td>16768.56</td>
<td>17019.18</td>
<td>16822.92</td>
</tr>
<tr>
<td>T=5</td>
<td>16358.32</td>
<td>16834.72</td>
<td>16823.96</td>
<td>17006.18</td>
<td>16796.24</td>
</tr>
<tr>
<td>T=6</td>
<td>16442</td>
<td>16806.36</td>
<td>16788.24</td>
<td>17027.28</td>
<td>16831.82</td>
</tr>
<tr>
<td>T=10</td>
<td>16472</td>
<td>16828.24</td>
<td>16812.18</td>
<td>17032.92</td>
<td>16847.4</td>
</tr>
<tr>
<td>T=20</td>
<td>16415.22</td>
<td>16826.18</td>
<td>16816.06</td>
<td>17047.44</td>
<td>16867.06</td>
</tr>
<tr>
<td>Our method</td>
<td>16608.16</td>
<td>16989.6</td>
<td>17003.7</td>
<td>17241.74</td>
<td>17063.72</td>
</tr>
<tr>
<td>Upper bound</td>
<td>18064.8</td>
<td>17518.8</td>
<td>17277</td>
<td>17495.4</td>
<td>17300.4</td>
</tr>
</tbody>
</table>

* Bold-faced numbers are the maximum for periodic consolidation.

TABLE 2
The total profit in Scenario 2.

<table>
<thead>
<tr>
<th>S</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>50</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure placement</td>
<td>16657.06</td>
<td>18447.62</td>
<td>20371.24</td>
<td>22705.54</td>
<td>24885.56</td>
</tr>
<tr>
<td>T=1</td>
<td>16645.16</td>
<td>18495.72</td>
<td>20373.46</td>
<td>22691.78</td>
<td>24622.6</td>
</tr>
<tr>
<td>T=2</td>
<td>16648.62</td>
<td>18452.74</td>
<td>20297.6</td>
<td>22595.84</td>
<td>24517.48</td>
</tr>
<tr>
<td>T=3</td>
<td>16636.7</td>
<td>18482.76</td>
<td>20400.44</td>
<td>22651.26</td>
<td>24624.98</td>
</tr>
<tr>
<td>T=5</td>
<td>16586.32</td>
<td>18511.26</td>
<td>20342.14</td>
<td>22691.44</td>
<td>24645.44</td>
</tr>
<tr>
<td>T=6</td>
<td>16598.16</td>
<td>18468.18</td>
<td>20300.54</td>
<td>22690.84</td>
<td>24584.08</td>
</tr>
<tr>
<td>T=10</td>
<td>16629.2</td>
<td>18510.22</td>
<td>20329.46</td>
<td>22694.28</td>
<td>24757.98</td>
</tr>
<tr>
<td>T=20</td>
<td>16603.42</td>
<td>18496.86</td>
<td>20329.18</td>
<td>22664.12</td>
<td>24552.22</td>
</tr>
<tr>
<td>Our method</td>
<td>16789.76</td>
<td>18677.68</td>
<td>20533.44</td>
<td>22821.92</td>
<td>24710.1</td>
</tr>
<tr>
<td>Upper bound</td>
<td>17846.4</td>
<td>20085</td>
<td>22815</td>
<td>26075.4</td>
<td>29117.4</td>
</tr>
</tbody>
</table>

6.4 The running time

Fig. 7(a) and (b) demonstrate the comparative results of the running time among the pure VM placement, the periodic consolidation with \( T = 1 \) (with the most frequent VM migration), and the proposed scheme, for Scenarios 1 and 2, respectively. In particular, our computer is configured with an Intel Core i5 2.3GHz CPU and 4GB RAM. The simulation results show that, though the pure VM placement has the lowest running time due to the lack of VM migrations, the proposed algorithm still effectively reduces the required time compared with the periodic consolidation (e.g., less than 27% of that of periodic consolidation). This result shows that the duration of the proposed defragmentation scheme is very acceptable. Finally, the running time of all schemes increases with the increase of \( \pi \) and \( S \).

6.5 The Impact of \( n' \)

In our previous discussions, we have demonstrated that the proposed PRD algorithm performs well in Scenario 2 even with inaccurate estimation of \( n' \). In this subsection, we further investigate the impact of inaccuracy in Scenario 1. In this experiment, we analyze the total profit, the average migration cost and the average service delay of our method with different \( n' \), \( n' \in \{50, 100, 150, 200, 250\} \), when \( \pi = 150 \). The results are shown in Table 3, where we can see that the total profit can be improved if the estimation is more accurate. In particular, higher profit can be achieved when \( n' = 150 \) and \( n' = 250 \). Nevertheless, by comparing Table 1 and Table 3, we can observe that our algorithm can still lead to a better profit than pure VM placement and periodic server consolidation (except for \( n' = 50 \)). On the
other hand, we observe that the average cost/service delay of our algorithm may increase with $n'$ in this experiment.

6.6 The Impact of $\ell_r$

Finally, since the service delay has the similar results to that of migration costs, we investigate the impact of $\ell_r$ on the average migration cost only. In Fig. 8, we illustrate the cost vs. $\ell_r$ for Scenarios 1 and 2. We can clearly observe that, with the increase of $\ell_r$, the cost of both server consolidation and our scheme increase. However, the cost of our scheme increases much slower than that of the periodic server consolidation.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have systematically investigated the resource fragmentation problem in cloud data centers that may limit the accommodation of dynamic VM demands in mobile cloud computing. In particular, we first identified the resource fragmentation problem due to fragmented resources in different servers. We then proposed a novel framework to minimize unnecessary VM migrations. Within the proposed framework, we formulated an optimization problem for resource defragmentation at a particular time epoch. We then developed an efficient heuristic algorithm that can obtain near-optimal results. To evaluate the proposed framework, we have also conducted extensive numerical results that demonstrate the advantages of our framework in practical scenarios.

In this paper, we have only considered the defragmentation of a single type of resources. In our future study, we will investigate the fragmentation problem with multidimensional resources, such as computing, memory, disk I/O, and network I/O. In addition, we will also investigate the defragmentation issue when VM requests are correlated.

REFERENCES


Weigang Hou received the Ph.D. degree in communication and information systems from Northeastern University, Shenyang, China, in 2013. He is currently an associate professor in College of Information Science and Engineering, Northeastern University, Shenyang, China. His research interests include green optical networks, cloud computing and network virtualization. He has published over 20 technical papers in the above areas.

Rui Zhang received his B.S. and M.S. degrees from the Department of Mathematics, Imperial College London, London, in 2010 and 2011, respectively. He is currently a Ph.D. student at the Department of Computer Science, City University of Hong Kong. His main research interest is cloud computing security.

Wen Qi received his B.E. degree from the Department of Automation, Nankai University in 2009, and the M.S. degree in computer science from the City University of Hong Kong in 2013. He is currently a Ph.D. student in the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing, networking, and security.

Dr. Kejie Lu (S’01-M’04-SM’07) received the B.S. and M.S. degrees in Telecommunications Engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively. He received the Ph.D degree in Electrical Engineering from the University of Texas at Dallas in 2003. In July 2005, he joined the Department of Electrical and Computer Engineering, University of Puerto Rico at Mayagüez, where he is currently an Associate Professor. Since January 2014, he has been an Oriental Scholar with the School of Computer Engineering, Shanghai University of Electric Power, Shanghai, China. His research interests include architecture and protocols design for computer and communication networks, performance analysis, network security, and wireless communications.

Dr. Jianping Wang is currently an associate professor in the Department of Computer Science at City University of Hong Kong. She received her BSc and MSc degrees from Nankai University in 1996 and 1999 respectively, and her Ph.D. degree from University of Texas at Dallas in 2003. Her research interests include Dependable Networking, Optical Networking, Service Oriented Wireless Sensor/Ad Hoc Networking.

Lei Guo is a Professor at Northeastern University, Shenyang, China. He received the Ph.D. degree from University of Electronic Science and Technology of China, China, in 2006. His research interests include communication networks, optical communications and wireless communications. He has published over 200 technical papers in the above areas on international journals and conferences, such as IEEE TCOM, IEEE TWC, IEEE/OSA JLT, IEEE/OSA JOCN, IEEE GLOBECOM, IEEE ICC, etc. Dr. Guo is a member of the IEEE and the OSA, and he is also a senior member of CIC. He is now serving as an editor for five international journals, such as Photonic Network Communications, etc.