A Profit-aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers

Wei Chen, Xiaoqiang Qiao, Jun Wei, Tao Huang
Institute of Software
Chinese Academy of Sciences
Beijing, China
{wchen, qiaoxiaoqiang, wj, tao}@otcaix.iscas.ac.cn

***Abstract*—As a rising application paradigm, cloud computing enables the resources to be virtualized and shared among applications. In a typical cloud computing scenario, customers, Service Providers (SP), and Platform Providers (PP) are independent participants, and they have their own objectives with different revenues and costs. From PPs' viewpoints, much research work reduced the costs by optimizing VM placement and deciding when and how to perform the VM migrations. However, some work ignored the fact that the balanced use of the multi-dimensional resources can affect overall resource utilization significantly. Furthermore, some work focuses on the selection of the VMs and the target servers without considering how to perform the reconfigurations. In this paper, with a comprehensive consideration of PPs' interests, we propose a framework to improve their profits by maximizing the resource utilization and reducing the reconfiguration costs. Firstly, we use the vector arithmetic to model the objective of balancing the multi-dimensional resources use and propose a VM deployment optimization method to maximize the resource utilization. Then a two-level runtime reconfiguration strategy, including local adjustment and VM parallel migration, is presented to reduce the VM migration and shorten the total migration time. Finally, we conduct some preliminary experiments, and the results show that our framework is effective in maximizing the resource utilization and reducing the costs of the runtime reconfiguration.**

Keywords-cloud computing; virtual machine; deployment optimization; runtime reconfiguration; migration

I. INTRODUCTION

Cloud computing [1] is popular as a rising application paradigm, where resources, including software, platform and infrastructure, are provided and shared as services. One important impetus of cloud computing is virtualization technology [2], which makes the resources (e.g. CPU, memory, etc.) as virtual ones and enables multiple applications to run on Virtual Machines (VMs) instead of Physical Machines (PMs). In a cloud computing environment, such as a virtualized data center, VM is the basic deployment and management unit. Cloud platform providers are responsible for various resource demands by determining *where* to place VMs and *how* to allocate the resources. The virtualization-based cloud computing can improve resource utilizations, scalabilities, flexibilities and availabilities of applications. Also it can provide good application isolations in multiple levels. Due to these advantages, large-scale distributed applications are preferred to be hosted in a cloud platform.

In a typical cloud computing scenario, customers, Service Providers (SP), and Platform Provider (PP) are independent participants. Customers purchase services from SPs. SPs maintain application services and rent platform resources in a *pay-per-use* way. PP offers computing resources in form of VMs, which have various resource capacities with corresponding charges. The stakeholders have their own objectives with different revenues and costs. As shown in Fig. 1, customers want *good services* (e.g. Performance, security, etc.) with *low charges*; SPs get incomes by *providing high-quality services* and paying computing resource rents; PP gets revenues from tenants (SPs) and affords the costs of maintaining the platform.

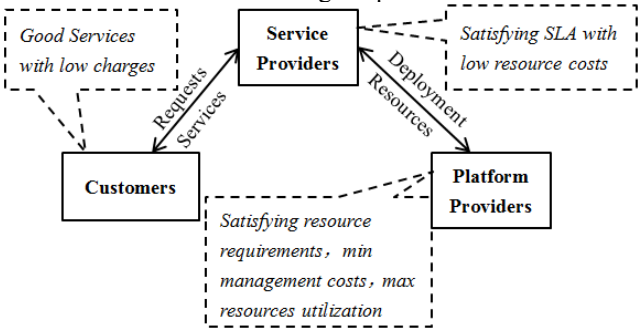


Figure 1. The stakeholders and their relations in cloud computing

For cloud platform providers, such as EC2 and NewServers, they can increase total profits by increasing the revenues and reducing the costs. However, once SPs declare their resource demands the revenue of PP is fixed. As a result, reducing costs becomes the only effective way to increase the profits. PPs' total costs come from several fields, which include physical resource, VM deployment and reconfiguration, power consumption and thermal management. Existing work reduced the costs by optimizing VM placement and deciding when and how to perform VM migrations.

However, there are some factors related to reducing PPs' costs are neglected by most of the existing work.

1) Balance of the multi-dimensional resources utilization. There are hundreds and thousands of applications running on a platform, and they are packaged into various VMs. These VMs demands different resources, typically including CPU, memory and network I/O. As defined in [17], *the balance of the multi-dimensional resources utilization means the resource in each dimension is utilized proportionally to the total amount.* If the administrators deploy VM improperly, some resources may

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.60

run out, while others remain a lot. Finally, the waste of resources in some dimensions leads to inefficient resource utilization, which generates much more resource costs.

2) VM Runtime reconfiguration is an important source of the platform management costs. PPs maintain applications' performance on their platform by adjusting the VM capacities and doing VM migrations. VM migration is the primary source of runtime reconfiguration cost due to its interferences on application performance, while VM capacity adjustment can be performed in a short time.

3) Different interests between initial deployment and runtime reconfiguration. During initial deployment, PPs have enough time to perform an effective optimization according to the resource demands from SPs. At runtime, the real-time requirements should be guaranteed by shortening the reconfiguration time. Furthermore, all application performance must not be degraded significantly otherwise the SLA between PPs and SPs may be violated.

Considering above factors comprehensively, we propose a VM deployment and reconfiguration optimization framework, which helps PPs increase their profits. We consider the balanced use of multi-dimensional resources and propose a VM deployment optimization method. Then a two-level runtime reconfiguration strategy, including time division multiplexing based local adjustment and parallel migration, is given to reduce the reconfiguration costs.

The rest of this paper is organized as follows. We present the motivation in Section II based on some observations. Problem analyses and formulations are introduced in Section III, and the overview of the framework is given in Section IV. The details of initial VM deployment and runtime reconfiguration are presented in section V and VI respectively. Experiments and evaluations are shown in section VII. We introduce the related work and give a final conclusion in section VIII and IX.

II. MOTIVITATION

We propose the VM deployment and reconfiguration optimization framework based on following observations.

1) SPs care about the resource provision and the costs they bear, and the locations of the applications are transparent to them. According to the demands and the workload, PPs can deploy VMs properly to get the max resource utilization without affecting SPs' profits.

2) As discussed in [17], it is not proper to represent the metric of the multi-dimensional resources utilization as a scalar. Mishra discussed *Sandpiper* [18] and *VectorDot* [19] to show the drawbacks in transforming a multi-dimensional metric into a scalar one [17]. Thus, the multi-dimensional resources should be modeled as a vector, and the utilization should be optimized based on the vector arithmetic.

3) With the fixed number of VM instances, runtime migration and local adjustment can deal with the fluctuating workload and resource demands. VM migration has to copy the memory states of the VM to a new server, and the VM's RAM and the available network bandwidth are decisive factors of the operation time. VM migrations use the network I/O in a preemptive way, which may degrade the performance of the applications, even violate the SLA. Therefore, the longer the migration, the larger the cost will be. In contrast, the local adjustment can be performed in a short time, and the cost can be neglected.

To increase PPs' profits with fixed revenues, we are motivated to optimize VM deployment and reconfiguration to reduce their costs from following aspects.

- Employ the balance of the multi-dimensional resources utilization as an optimization objective, and model the objective as a vector based metric. Maximize the resources utilization based on a multi-objective optimization method.
- According to the resource demands and the workload features, adjust VMs to avoid unnecessary VM migrations based on *time division multiplexing*.
- If migration is inevitable, considering the available network bandwidth and VM size, reduce the total time by moving multiple VMs in a parallel way.

III. PROBLEM ANALYSIS AND FORMULATIONS

A. Primary Factors Analysis

PPs' revenues are fixed once SPs declare their demands, and PPs have to increase the total profits by minimizing their costs. PPs' costs come from different aspects, including resource utilization [17], power consumption [14], thermal management [12][13], VM reconfiguration [21] and the penalties of SLA violation [8]. In this paper, we concentrate on the costs of the resource utilization and the VM reconfiguration and get our optimization objectives, including: a) maximizing the resource utilization and b) minimizing the operation costs.

We analyze the primary factors to our optimization objectives. Total Resource Capacity (TRC) and Residual Resource Capacity (RRC) have significant effects on VM deployments. TRC affects the initial VM placements, and RRC determines how many additional VMs can be deployed. In addition, the smaller the RRC, the better the resource utilization is. Residual Network Bandwidth (RNB) affects VM reconfiguration costs. Other VMs on the same server are affected due to a lot of network I/O is utilized by VM migration, and the longer the migration, the bigger the effect is. Sufficient RNB can make the VM migration faster and avoid the penalties of SLA violation. Application features, including Application Type (AT), Resource Capacity Requirement (RCR) and Request Arrival Rate (RAR), are important factors. AT includes computing intensive, communication intensive, data intensive and others. Different type applications have various resource demands in each dimension. For example, a computing intensive application requests much more CPU, and a communication intensive one requires more network I/O. Due to RCR and AT, different VM deployments in a server result in various resource utilizations. RAR affects the resource utilization positively. When RAR increases, application workload will increase and more resources are required.

Based on the above analysis, PPs' deployment optimization problem we consider can be formulated.

B. Problem Formulation

We suppose there are K applications partitioned and packaged into M ($K \le M$) VMs. These VMs are going to be deployed in a platform constituted of N PMs. After running for some time, L VMs are overloaded. Therefore, the VMs must be reconfigured to maintain the performance. TABLE I lists the symbols used in the rest of this paper.

TABLE I. SYMBOLS USED IN THE PROBLEM FORMULATION

K	Number of applications
M	Number of VMs
N	Number of physical servers
L	Number of VMs in overloaded state
$D_{M\times N}$	VM allocation Matrix
i	ith VM
j	jth server
R_c	Total CPU capacity of a server
R_m	Total memory capacity of a server
R_{io}	Total I/O capacity of a server
λ_c	Scale factor of CPU
λ_m	Scale factor of memory
λ_{io}	Scale factor of network I/O
C_{mig}	VM migration cost
mem_{vm}	VM RAM size
$\overrightarrow{RCV}$	Residual capacity vector
$\overrightarrow{TCV}$	Total capacity vector
$\overrightarrow{RRV}$	Resource requirement vector
$\overrightarrow{UCV}$	Utilized capacity vector

We model TRC as a vector $\overrightarrow{TCV}$, which represents the total multi-dimensional resources, and each component stands for a resource. The vector is instantiated as $\overrightarrow{TCV} = <\lambda_c R_c, \lambda_m R_m, \lambda_{io} R_{io}>$, which means CPU ($R_c$), memory ($R_m$) and network I/O ($R_{io}$) are concerned in this paper. Excessive use of resources (e.g. CPU) can lead to high power consumption and local hotspot [15] [16], which can increase management costs. Therefore, λ_c, λ_m and λ_{io} are scale factors to restrict the total resource can be used, whose values are in range *(0.5,1]* and are always approximate to *1*. Similar to $\overrightarrow{TCV}$, Residual Capacity Vector ($\overrightarrow{RCV}$), Resource Requirement Vector ($\overrightarrow{RRV}$) and Utilized Capacity Vector ($\overrightarrow{UCV}$) are also represented. Based on these vectors, we define the balance of the multi-dimensional resources utilization.

As presented in [17], the balance of resources utilization is a state, where each resource is used proportionally to its total amount. For example, the utilizations of 50% CPU, 50% memory and 50% network I/O are much balance than the one of 50% CPU, 40% memory and 10% network I/O. In a balance state, it can be avoided that some resources run out while others remain a lot. We use the angle between $\overrightarrow{UCV}$ and $\overrightarrow{TCV}$ to evaluate the balance degree. The greater the angle, the less the balance will be.

Total migration time of VMs is usually an important metric for migration cost [22]. The longer the migration, the higher the cost will be. We formulate the runtime VM reconfiguration cost as:

$$C_{mig} = \frac{mem_{vm}}{RNB}$$

Where C_{mig} is the migration cost, represented by the time of transmitting a VM from the current server to another one.

Our optimization objectives can be formulated as:

- Max $\min_{j\in 1..N} \frac{\overrightarrow{UCV_j} \cdot \overrightarrow{TCV_j}}{\left|\overrightarrow{UCV_j}\right| \cdot \left|\overrightarrow{TCV_j}\right|}$
- Max $1/S_r$
- Min $\sum_{i=1}^{L} C_{mig,i}$

The first formula is the objective of balancing the multi-dimensional resource use. The formula represents the cosine of the angle between $\overrightarrow{UCV}$ and $\overrightarrow{TCV}$. We try to minimize the max angle among all servers. If a $\overrightarrow{UCV}$ exactly aligns with a $\overrightarrow{TCV}$, the server reaches the most balance of the resource utilization. The second formula is the objective of maximizing the resource utilization, where S_r is the number of servers not in the idle state. This formula means trying to use the minimum number of servers to host these VMs. The last one is to minimize the reconfiguration cost when L overloaded VMs exist.

There are some constraints should not be violated when optimizing VM deployment and reconfiguration.

- $\forall j \in N \quad \sum_{i\in M} RRV_{mem} D_{i,j} \le \lambda_{mem} TCV_{mem}$
- $\forall j \in N \quad \sum_{i\in M} RRV_{cpu} D_{i,j} \le \lambda_{cpu} TCV_{cpu}$
- $\forall j \in N \quad \sum_{i\in M} RRV_{io} D_{i,j} \le \lambda_{io} TCV_{io}$
- $\sum_{j=1}^{N} d_{ij} = 1, i \in [1 \quad M]$

The first three constrain the total demands not to exceed the server capacities, and the last one ensures each VM is allocated to one and only one server.

IV. THE FRAMEWORK OVERVIEW

As shown in Fig.2, there are four primary components in the framework.

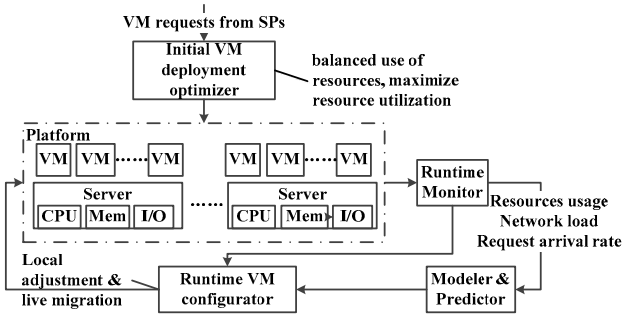


Figure 2. Overview of the framework

Initial VM Deployment Optimizer (IVDO) focuses on the static VM deployment, whose responsibilities are balancing and maximizing resource utilization. This component realizes the optimization method based on Grouping Genetic Algorithm (G-GA) [20] to achieve the high scalability and good performance.

Runtime Monitor (RM) gets the runtime information, including resource usage, network load, and request arrival

rate. All information is sampled periodically without affecting application performance significantly.

Based on the formulas presented in Section III, Modeler and Predictor (MP) use data from RM to calculate the objective values and predict the future state.

Runtime VM Configurator (RVC) decides *when* and *how* to reconfigure the VMs. To reduce the runtime reconfiguration costs, RVC is responsible for reducing the number and the total time of VM migrations.

IVDO and RVC are the two primary components used to reduce PPs' costs in our framework. We introduce the details of the methods used in IVDO and RVC in the next sections.

V. INITIAL VM DEPLOYMENT OPTIMIZATION

VM deployment is a multi-objective optimization problem, and GA has been proven having the best scalability and performance among several methods, including linear programming, nonlinear programming, and greedy [3]. We use G-GA, an extension of GA, to solve this problem.

In G-GA based VM deployment optimization, an individual has *N* genes representing the servers, which is encoded by a group of VMs on the server. For example, individual *({1, 3, 4}, {2, 5}… {9 ... M})* is a solution to the problem of allocating *M* VMs to *N* servers.

The G-GA based algorithm in IVDO is shown in Fig.3. There are three aspects to be addressed: 1) *initial individual generation*, 2) *crossover* and 3) *mutation*.

1) Initial individual generation

We propose some heuristics to generate initial individuals other than randomly.

The same VM instances must be allocated to different servers. An application is usually partitioned into several parts according to its architecture. For each part, the SP may demand multiple instances to ensure the application performance, availability and other QoS. Therefore, the same VM instances must be distributed to different servers to avoid resource contention and QoS degradation.

The different VM instances of the same application are tried to be deployed on a same server. The different VM instances constituting an application may communicate much more frequently (e.g. the function call and the database access). Placing such VM instances on the same server helps to reduce the network workload.

All heuristics must be used based on the constraints introduced in Section III.

2) Ranking based crossover

In G-GA, crossover generates the offspring by exchanging partial genes of two parents. We propose a ranking based crossover based on *Pareto optimum* [4] to make this method has better convergence.

Two objectives, the resource utilization and the balance of the multi-dimensional resources utilization, are as metrics to evaluate a gene is whether Pareto optimal or not. All genes of parents are ranked and grouped into multi-sets. Genes in the same set are *non-dominated* to one another; genes in different sets belong to the different Pareto optimal levels. *When doing the crossover, we select genes from the sets in descending order to make the offspring evolve toward the optimal direction.*

After crossover, some VMs may appear twice, and others may be missing in the resulting offspring. To ensure the integrity, the groups containing duplicated VMs in the second parent are eliminated, and the missing ones are re-inserted in.

Algorithm: G-GA based VM deployment optimization

Input: population size (K), number of generations (G), crossover rate (R_c), mutation rate (R_m), VM demands (L_{RRV}), Server capacities (L_{TCV}), best-fit threshold (T_b), worst-fit threshold (T_w), termination threshold (T_t)
Output: best K solutions
GGABasedOpt (K, G, R_c, R_m, L_{RRV}, L_{TCV}, T_b, T_w, T_t)

```
1   List optimal_d,   offspring;
2   optimal_d = initGeneration(K, L_RRV, L_TCV); //generate the
3   initial k individuals based on the heuristics
    for (int i=0;i<G;i++) {
4     for (int j=0; j<K* R_c;j++) {
5       parents (x,y)= parent_selection(optimal_d);
6       initial_offspring = ranking_based_crossorver(x,y);
7   //crossover based on praetor optimal heuristics
        initial_offspring = insertion(initial_offspring, T_b, T_w,);
8       offspring.add( initial_offspring);
9     }
10    for (int l=0; l<K* R_m;l++)
11      parent x = parent_selection(optimal_d); //select parent
12      initial_offspring = mutation(x);
13      offspring.add(initial_offspring);
14    }
15    difference = get_worst_difference (optimal_d,
16    offspring);//get the worst difference
      if (difference < T_t) {
17      break;
18    } else {
19      optimal_d = evaluate_select(optimal_d, offspring, K);
20    }
21  }
22  return optimal_d;
```

Figure 3. Primary Algorithm of IVDO

3) Mutation and missing VMs re-insertion

The target server selection is the key to the mutation and the VM reinsertion. *Best-fit, worst-fit* and *random* are common local search heuristics, and we use them as following.

According to $\overrightarrow{RCV}$ and $\overrightarrow{RRV}$, we first select the best-fit server with available resources to minimize the resource fragments. If there are no suitable targets, the worst-fit one is selected to ensure the residual capacities are still enough to host the other VMs. We select one target randomly when a suitable server is not found by previous two heuristics.

VI. RUNTIME VM RECONFIGURATION

To reduce the reconfiguration costs mainly from VM migration (other operations costs, e.g. local VM capacity adjustments and request distribution, are negligible.), we propose a two-level reconfiguration method, including local adjustment and VM migration.

A. Local Adjustment

Local adjustment is the preferred operation due to its negligible cost. We use *resource borrowing* and *workload distribution* together to satisfy the demands of the overloaded VMs. The method is proposed based on following intuitions.

1) The projected resource demands of the applications are specified based on the off-line estimations with their peak workload. In practice, the workload is time varying and the applications will not be always in high-loaded states.

2) The multiple instances of an application are deployed on different servers, and it is feasible to adjust the resource demands of each instance by distributing the different workload to them.

Therefore, according to the predicted workload and the resource utilizations, can adjust the VMs capacities based on *time division multiplexing*. The method means that, in a physical server, resources can be borrowed from light-loaded VMs and reallocated to the overloaded ones according to the VMs' current states and their workload features. As shown in Fig.4, the local adjustment process is as following.

For an overloaded application with multiple instances distributed on the different servers, we firstly predict: 1) the total resource demands and the duration of its peak workload, 2) the resource utilization trends of the other VMs on the same servers, and 3) the next peak workload of the other VMs. Secondly, we estimate the total resources can be borrowed from the light-loaded VMs. Thirdly, we adjust the resource capacities of VMs on these servers. We *borrow* resources from the light-loaded VMs and re-allocated to the overloaded ones, where a *weighted based sharing* heuristic is used. Among the multiple instances in these servers, we then re-distribute the total workload of the overloaded application. As a result, the previous overloaded VM instances can afford more workload proportionally to the additional resources they obtain. Finally, when the workload decreases, all resources borrowed are returned to the original owners.

In this process, *weighted based sharing* heuristic is: according to $\overrightarrow{RCV}$ and the predicted start time of the peak load, the light-loaded VMs in the same server are ranked with different weightiness. The VM with lower weightiness means it has more resources and the longer time to its peak workload, and such VMs are selected at first.

B. Parallel VM Migration

The local adjustment reduces VM migrations, but it cannot deal with all cases, particularly the resource contentions. In such situations, the VM migrations are necessary. There is some existing work addressing the problem of how to select and move a VM each time. We consider the scenario that there are multiple VMs should be migrated, which is common in a large-scale cloud platform.

Parallel migration has been proven as an effective way to shorten the total time if there are multiple VM instances to be migrated [9]. The experiment in [9] shows that "*parallelizing the migration of two VMs with different source nodes and different target nodes would shorten the total migration time significantly*". On the other hand, if we simultaneously move multiple VMs from a same server, many network I/O will be utilized, which may affect other VMs significantly due to the lack of the resources.

We propose a VM parallel migration method considering two aspects: 1) *which VMs should be migrated*, and 2) *how to migrate the VMs in a parallel way*.

1) Determining the VMs to be migrated

The following factors should be considered to decide which VMs to be migrated: 1) UCV_i – the Resource utilization in the *i*th dimension, which is the one that VMs contend for; 2) mem_{vm} – the VM RAM size; 3) *R - Ratio* of UCV_i to mem_{vm} of the VM, if the type of UCV_i is memory, $1/mem_{vm}$ equals to *R*. *R* with a bigger value means more resources are occupied proportionally to the unit RAM.

Algorithm: Local adjustment of VM

```
    Input: number of overloaded VM (L), number of
    corresponding server (P, P≤L),
    Output: local adjustment solution
    LocalAdjustment (L,P) {
1    List vm_demands,  peak_duration,  servers_ucv;
2    for (int i=0;i<L;i++) {
3       vm_info = estimate_resource_demand (i);
4       peak_dur = estimate_peak_duration(i);
5         vm_demands.add(vm_info);
6         peak_duration.add(peak_dur);
7    }
8    servers_ucv = get_server_ucv (P);
9    List res_borrow_list;
10   for (int j=0;j<P;j++) {
11      int otherVM = getVMnumber (j);
12      List single_server_borrow_res;
13      for (int k=0;k<otherVM;k++){
14         borrow_res = weighted_based_sharing (k);
15         single_server_borrow_res. add( borrow_res );
16      }
17      res_borrow_list.add( single_server_borrow_res);
18   }
19   List vm_after_adj;
20   for (int l=0;l<L;l++) {
21      new_capacity = re_allocate( res_borrow_list );
22      vm_after_adj.add ( new_capacity);
23   }
24   request_distribute( vm_after_adj);
25   return;
26 }
```

Figure 4. Local adjustment algorithm of RVC

Among the VMs contending for the *ith* type resource, we first decide the *minimum number* of VMs having to be moved. Then we select the VMs with the maximum sum of *R*. For example, within a server *s*, there are four VMs (*a, b, c* and *d*) contending for network I/O, and at least two VMs should be migrated. The VM set can be *{a, b}* or *{b, c}*, and we select the first set since the sum of R of which is bigger.

2) Migrating the VMs in a parallel way

To make tradeoffs between shortening migration time and interfering with the other VMs' performance, we prefer to migrate VMs from the different source nodes to the different target nodes simultaneously. If there are at least two VMs to be moved within a server, the VMs should be moved

one by one. After selecting the VMs, the target servers must be identified. In this step, $\overline{RCV}$ and *RNB* are two decisive factors. To parallelize the migrations, we select the targets from those having no resource contentions. The parallel migration is proposed as following.

Firstly, the servers having VMs to be moved are grouped as source nodes set *S*. Secondly, for each VM *i* to be migrated, we find the servers can host it as candidates set T_i, and the servers in T_i are not belonged to *S*. Thirdly, the union set *T* of the all sets T_i is as the total candidate set of the targets. Then, with *S* and *T*, we get the edges between the nodes in these two sets according to the VMs and their candidate targets. The weight of each edge denotes the relative transmission rate decided by *RNB and* VM mem_{vm}. Based on this problem formulation, the parallel VM migration can be mapped to the *max matching problem of the bipartite graph* [5]. Furthermore, if there is more than one VM to be moved in a same server, the migration is made *one VM per server* simultaneously. The migration process is as following.

Initially, we find the node set can be matched from the source to the target, and the total time is determined by the slowest migration. If there are still some VMs to be moved, we find the proper transmission paths based on the edge weight and the targets' free time relative to the longest one in the previous migration. Therefore, some migrations in the next iteration can be performed before the longest operation finishes. After a migration completes, the allocation matrix $D_{M\times N}$ and the nodes sets *S* and *T* are updated logically to evaluate whether there are VMs to be moved or not. This process iterates until there are no VMs to be moved.

VII. Experiments and Evaluations

A. Initial VM Deployment Optimization

We evaluate the effectiveness of our initial VM deployment method and compare it to another one used in *sandpiper* [18] to show the advantage in resource utilization.

In this experiment, we simulate the initial VM deployment. The VM template capacities and application demands are distributed in the following sets: CPU-{0.25*2.4, 0.5*2.4, 1*2.4, 1.25*2.4, 1.5*2.4, 2.0*2.4, 3*2.4, 4*2.4}, memory-{0.5, 1, 1.5, 2.0, 3.0, 4.0, 6.0}, network I/O-{4, 6, 10, 15, 20}. Physical server's $\overline{TCV}$ is {16*2.4GHZ, 24GB, 100M}. VM templates and application instances are created as the combination of elements in the resource sets. The problem size is varied by changing the numbers of applications, VM templates, and servers.

The simulated experiment results are shown in Fig.5, where the x-axis denotes the number of the servers without applications and the y-axis is the balance of the multi-dimensional resources utilization represented by the minimal cosine value of the angle between $\overline{UCV}$ and $\overline{TCV}$. We can see that, in most conditions, our method makes the VM deployments much more balanced in multi-dimensional resource utilization among the servers. Moreover, due to the efficient resource utilization, more servers can be saved to host more applications.

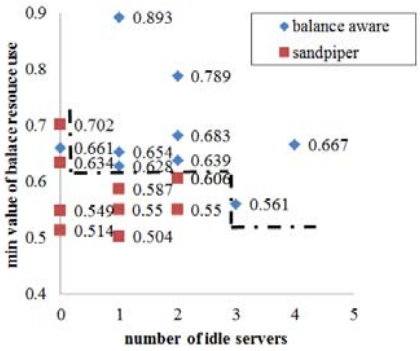


Figure 5. Experimental results of the initial VM deployment

B. Runtime VM Reconfiguration

We set up a VM-based computing platform constituted of several server nodes. Each node is configured with quad-core 2.4GHz processors and 8G RAM, and these nodes are connected with gigabit Ethernet.

1) Local Adjustment

We employ three types of applications, CPU-intensive (CI), Memory-intensive (MI) and Network I/O-intensive (NI), with multiple instances. Multiple VM templates are provided and allocated to these applications. TABLE II, III, and IV depict the servers' uses, VM template configurations and application instances respectively.

TABLE II. Use of Servers

ID	Use
S_1	File system for VM migration
S_2	Client workload generator for applications
S_3	Request router, distributing client requests
S_4-S_7	Hosting applications packaged into VM

TABLE III. VM Template

ID	Name	Configuration
V_1	Common	0.5*2.4GHZ, 1GB RAM, 10M I/O
V_2	High-CPU	2*2.4GHZ, 1.5GB RAM, 20M I/O
V_3	High-Mem	1*2.4GHZ, 3GB RAM, 20M I/O
V_4	High-I/O	1*2.4GHZ, 1GB RAM, 30M I/O

TABLE IV. Application Instances and Their Allocations

App. Type	App ID	VM Template	Instance number	Server ID
CPU-Intensive	CI-1	V_2	2	S_4, S_5
	CI-2	V_2	2	S_4, S_5
Mem-Intensive	MI-1	V_3	2	S_4, S_5
I/O-Intensive	NI-1	V_4	2	S_4, S_5

Among the applications, CI-1 and CI-2 need much CPU resource to support their tasks, including Pi calculation, k-means and other complex computations. MI-1 is realized to do intensive message transmission. NI-1 needs large amount of memory to cache data. All these applications have two instances, and they are distributed on different servers. There are also some other applications running on S6 and S7 to make them not in the idle state.

We conduct the experiment with the following steps:

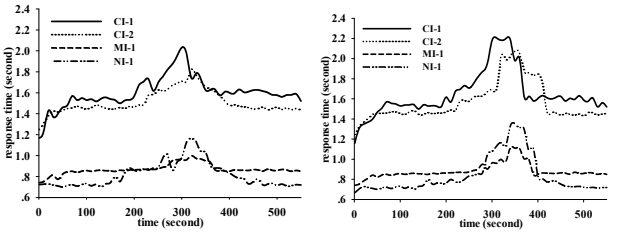
- Make applications run for a period of time and generate the peak workload for applications at the different time with different intervals.

- Increase the workload of CI-1, 2 and make them in the overloaded state. Now, local adjustment is used to configure the resource allocations among these applications and provide more CPU resources to overloaded ones.

To evaluate the effectiveness, we use local adjustment and the VM migration to reconfigure the platform respectively, and compare their effects to applications in terms of performance.

Experimental results are shown in Fig.6. We use application instances hosted on S4 as representatives. Figure 6(a) shows the result of local adjustment, and Fig.6 (b) is the result of migrating CI-1 to S6 and S7. We observe that, the application performance degradations during local adjustment are smaller than that during VM migration, and the time of performance degradations is also shorter.

The local adjustment avoids the unnecessary VM migrations by borrowing spare resource from other instances, and it spends less time during runtime configuration. In contrast, although VM migration can solve the problem, it needs much more time to do VM transmission, which generates significant interferences with the other applications. Meanwhile, compared with the memory intensive application, NI-1 receives more interference when doing the VM migration because the network I/O is occupied by the operation.



(a) Local adjustment (b) Reconfigure using VM migration

Figure 6. Reconfiguration when some VMs are overloaded

2) VM Parallel Migration

We conduct another experiment to evaluate the VM parallel migration method on this platform. Firstly, we place several VMs configured with 2.4GHz CPU resource and various RAM on five nodes, S1 to S5. Secondly, we make these nodes contend for different resources (2 for network IO, 2 for CPU and 1 for memory) by increasing their workload. Then, another five nodes, T1 to T5, are used as targets to host VMs being migrated. According to each $\overrightarrow{RCV}$ of these nodes, some VM instances on servers S1 to S5 are selected and moved to different target nodes. The experiment information is shown in TABLE V.

TABLE V. EXPERIMENT INFORMATION

Source Node	Contention	VM instances	Target Node
S_1	Network I/O	S_{11},S_{12},S_{13},S_{14}	T_1, T_2, T_3
S_2	Network I/O	S_{21},S_{22},S_{23},S_{24}	T_1, T_4
S_3	CPU	S_{31},S_{32},S_{33},S_{34}	T_4, T_5
S_4	CPU	S_{41},S_{42},S_{43},S_{44}	T_2, T_5
S_5	Memory	S_{51},S_{52},S_{53}	T_4, T_5

We employ different methods to select and move VMs. These methods are:

1) *Our Parallel Migration (OPM)*: as introduced in Section VI, we consider the multiple factors and use *bipartite graph matching* algorithms based parallel migration.

2) *Random VM selection and migration (RSM)*: select VMs to migrate without concerning the factors we proposed, and move these instances in parallel.

3) *Simultaneous VM migration (SVM)*: migrate all selected VMs simultaneously even there are VMs in the same source nodes.

TABLE VI. EXPERIMENT RESULTS

method	VM migration operation	Time (s)
OPM	S_{14}->T_3, S_{24}->T_1, S_{32}->T_5, S_{43}->T_2, S_{53}->T_4	57.74
	S_{12}->T_2, S_{21}->T_1, S_{34}->T_4, S_{41}->T_5	
RSM	S_{11}->T_3, S_{22}->T_1, S_{33}->T_5, S_{42}->T_2, S_{51}->T_4	72.96
	S_{13}->T_2, S_{23}->T_1, S_{31}->T_4, S_{44}->T_5	
SVM	$S_{12,13}$->T_3, $S_{24,23}$->T_1, $S_{34,31}$->T_5, $S_{43,41}$->T_2, S_{52}->T_4	64.81

The migration results and time costs are shown in TABLE VI. We find that OPM and RSM perform parallel migrations in two iterations. Without considering the factors of selecting VMs, the VMs to be moved in RSM are not the most appropriate ones, which need much more network bandwidth to finish the migrations. As a result, RSM performs the longest time among these experiments. Although SVM has better performance than RSM, all instances are migrated simultaneously, and much of the network I/O is taken up. We find that in the experiment with SVM the other VM instances receive more interference to their performance due to the lack of the network I/O.

VIII. RELATED WORK

There is some work devoted to addressing the problems of VM deployment, placement and runtime reconfiguration with different objectives.

From SPs' viewpoint, work in [10] and [11] guided VM template selection and runtime reconfiguration according to resource price, reconfiguration cost and the fluctuating workload. Reference [10] used Linear Programming (LP) to solve the problem. Work in [11] took both resource price and the operation cost into account and adapted the VMs in advance based on the performance predictions.

The research work from PPs' viewpoint is common. Work in [18] [19] only considered the resource utilizations and the workload, using classical *bin-packing*, *greedy algorithm* and other methods to get solutions. Recently, some factors relating to providers' costs (e.g. power consumption, thermal dissipation and operation costs, etc.) were considered. Temperature-aware workload placements were presented in [12] and [13]. D. Jayasinghe [6] found the infrastructure structure affected the performance, availability and communication, then he proposed a hierarchical approach to guide VM placement. Y. Kang [7] focused on the factors of user physical distribution and network transmission, and he mapped VM placement to the *k-median* and *max k-cover* problem to improve end users' experiences.

Work in [17] is similar to our method in modeling the multi-dimensional resources. The authors used a normalized resource cube to represent the 3-dimensional resources and got a regular hexagon by projecting the cube on a plane

perpendicular to its principal diagonal. They proposed a set of algorithms to select the most suitable PM and perform the VM placement based on the regular hexagon. Unlikely, we consider the balanced use of multi-dimensional resources just in initial deployment and use the cosine of the angle between $\overrightarrow{UCV}$ and $\overrightarrow{TCV}$ as an objective model. Our method can be extended and used in more than three-dimensional resources utilization, while work in [17] is limited in their cube model of the specific resource.

J. Xu [15][16] simultaneously considered workload, power consumption and thermal management when performing initial deployment and runtime reconfiguration. Previous work only considers *when* and *where* to reconfigure single VM each time, while our work considers *how* to adjust and move multiple VMs at runtime. Our ultimate goal is to increase PPs' profits by maximizing the resource utilization and reducing the reconfiguration costs.

IX. Conclusion and Future Works

We address the problem of the VM deployment and reconfiguration from PPs' viewpoint. We identify the different concerns of PPs in initial deployment and runtime reconfiguration. Based on the interests, the methods are proposed to reduce the costs in both phases. The following contributions are made to increase PPs' profits.

- We consider the balance of the multi-dimensional resources utilization and propose an optimization method to maximize the resource utilization.
- According to the features of resource demands and workload, a *time division multiplexing* based VM local adjustment strategy is proposed to reduce the number of the VM migration.
- We realize a VM *parallel migration* to reduce the total migration time based on the *max matching problem of the bipartite graph*.

However, there are still some limitations in our work, including: 1) the prediction techniques may have deviations to the facts, which affect the runtime reconfiguration decisions; 2) the interferences between VMs on a same server are ignored. In future work, we will handle these limitations by employing the much more accurate prediction techniques and taking the interferences between VMs into account.

Acknowledgment

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2009CB320704, the National High Technology Research and Development Program of China under Grant No. 2012AA011204, the National Natural Science Foundation of China under Grant No. 61173003, 61100065.

References

[1] M. Armbrust, et al. "Above the clouds: A Berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, U.C. Berkeley, 2009.

[2] J. E. Smith, and R. Nair, "The Architecture of Virtual Machines," Computer, 38(5):32-38, 2005.

[3] S. Malek, et al, "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture," IEEE Trans. On Software Engineering, 2011.

[4] P. Godfrey, R. Shipley, J. Gryz, "Algorithms and Analyses for Maximal Vector Computation," The International Journal on Very Large Data Bases, vol. 16, Issue 1, 2007.

[5] D. West, Introduction to Graph Theory. Prentice Hall, 2007.

[6] D. Jayasinghe, et al, "Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-aware Virtual Machine Placement," in Proceedings of 8th International Conference on Service Computing, 2011.

[7] Y. Kang, et al, "A User Experience-based Cloud Service Redeployment Mechanism," in Proceedings of 4th International Conference on Cloud Computing, 2011.

[8] N. Bobroff, et al, "Dynamic placement of virtual machines for managing SLA violations," in Proceedings of 10th IFIP/IEEE International Symposium on Integrated network management, 2007.

[9] H. Jin, L. Deng, et al., "Dynamic Processor Resource Configuration in Virtualized Environments," in Proceedings of 8th IEEE International Conference on Service Computing, 2011.

[10] U. Sharma, P. Shenoy, et al, "A Cost-Aware Elasticity Provisioning System for Cloud," in Proceedings of 31st International Conference on Distributed Computing Systems, 2011.

[11] H. Wu, WB. Zhang, J. Wei, et al, "A Benefit-Aware On-Demand Provisioning Approach for Cloud Computing," in Proceedings of the 3rd Asia-Pacific Symposium on Internetware, 2011.

[12] L. Ramos and R. Bianchini, "C-Oracle: Predictive Thermal Management for Data Centers," in Proceedings of the 14th International Symposium on High-Performance Computer Architecture (HPCA 14), 2008

[13] Q. Tang, S. Gupta and G. Varsamopoulos, "Energy-Efficient, Thermal-Aware Task Scheduling for Homogeneous, High Performance Computing Data Centers: A Cyber-Physical Approach," in Trans. On Parallel and Distributed Systems, Spec. Issue on Power-Aware Parallel and Distributed Systems, 2008.

[14] G. Jung, M. Hiltunen, et al, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures," in Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS), 2010.

[15] J. Xu and J. Fortes, "Multi-objective Virtual Machine Placement in Virtualized Data Center Environments," Green Computing and Communications (GreenCom), IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom) , 2010.

[16] J. Xu and J. Fortes, "A Multi-objective Approach to Virtual Machine Management in Datacenters," in Proceedings of 8th International Conference on Autonomic Computing, 2011.

[17] M. Mishra and A. Sahoo, "On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach," in Proceedings of 4th International Conference on Cloud Computing, 2011.

[18] T. Wood, et al, "Black-Box and Gray-Box Strategies for Virtual Machine Migration," in Proceedings the 4th USENIX Conference on Networked Systems Design and Implementation, 2007.

[19] A. Singh, et al, "Server-Storage virtualization: Integration and Load Balancing in Data Centers," in Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008.

[20] E. Falkenauer, "A Hybrid Grouping Genetic Algorithm for Bin Packing," Journal of Heurisitcs, vol. 2, Number 1, 1996.

[21] M. Tarighi, et al, "A New Model for Virtual Machine Migration in Virtualized Cluster Server Based on Fuzzy Decision Making," Journal of Telecommunications, vol. 1, issue 1, pp. 40-51, Feb. 2010.

[22] F. Hermenier, X. Lorca, et al, "Entropy: a consolidation manager for clusters," in Proceedings of the ACM/Usenix International Conference on Virtual Execution Environments (VEE), 2009