Virtual Machine Placement in Cloud Systems through Migration Process

Kangkang Li, Huanyang Zheng, Jie Wu, and Xiaojiang Du
Department of Computer and Information Sciences
Temple University, Philadelphia, PA, 19122
Email: {kang.kang.li, huanyang.zheng, jiewu, dux}@temple.edu

***Abstract*—Cloud computing is an emerging technology that greatly shapes our lives, where users run their jobs on virtual machines (VMs) on physical machines (PMs) provided by a cloud service provider, saving the investment in upfront infrastructures. Due to the heterogeneity of various jobs, different VMs on the same PMs could have different job completion times. Meanwhile, the PMs are also heterogeneous. Therefore, different VM placements have different job completion times, and our objective is to minimize the total job completion time of the input VM requests through a reasonable VM placement schedule. This problem is NP-hard, since it can be reduced to a knapsack problem. We propose an off-line VM placement method through an emulated VM migration process, while the on-line VM placement is solved by a real VM migration process. The migration algorithm is a heuristic approach, in which we place the VM to its best PM directly, if this PM has enough capacity. Otherwise, we migrate another VM from this PM to accommodate the new VM, if a pre-specified migration constraint is satisfied. Furthermore, we study a hybrid scheme where a batch is employed to accept upcoming VMs for the on-line scenario. Evaluation results validate the high efficiency of the proposed algorithms.**

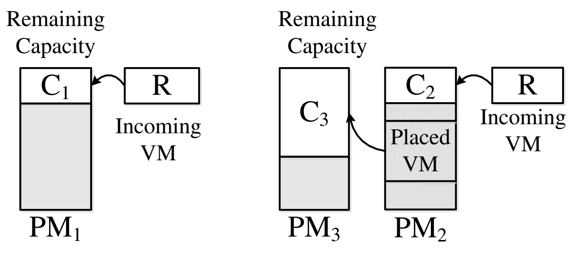
***Index Terms*—VM migration, job completion time, off-line, on-line, VM placement.**

I. Introduction

Nowadays, cloud computing is an emerging technology that greatly shapes our lives. With the large pools of computing and storage resources provided by cloud providers, many companies can rent these resources and run their jobs on virtual machines (VMs), saving the investment in upfront infrastructures. In particular, the technique of VM migration [1, 2] enables us to move a VM from one physical host to another. This spatial flexibility is effective in server consolidation, power consumption saving, and so forth.

Obviously, VMs are different among their resource demands and their running jobs. Due to this heterogeneity, different VMs running on the same PM could have different job completion times. Moreover, in most circumstances, the host PMs are also heterogeneous, such as different CPU architectures, different OS, differing amounts of memory, storage, etc. Hence, the same VM placed on different PMs would have different completion times, and our objective is to minimize the total job completion time of the input VM requests through a reasonable VM placement schedule. This problem is NP-hard, since it can be reduced to a knapsack problem.

VM migration is an efficient tool for resource provisioning, by dynamically rearranging the previous placement. For



(a) Direct placement (b) Migration-based placement

Fig. 1. An illustration of the VM placement

instance, when all PMs are heavily loaded without enough capacity to place the new VM, through migrating one running VM to another host, we could save space to accept the incoming VM. In this paper, we consider both off-line and on-line VM placement problems with the objective of minimizing the total job completion time.

Under the off-line scenario, the information of input VMs are known a priori. Therefore, we propose a VM placement algorithm through an *emulated* VM migration process, in which we place the VMs one-by-one. Suppose that there is an incoming VM waiting for placement in a cloud system with three PMs: PM_1, PM_2, PM_3. The job completion times of the incoming VM placed on PM_1, PM_2, PM_3 are, respectively, t_1, t_2, t_3. As shown in Fig. 1(a), for the incoming VM, we could directly place it in PM_1, which has enough remaining capacity to support it. Then, the total completion time will increase by t_1. However, this option might not optimal. Suppose that placing the VM in PM_2 has the minimal completion time t_2 (among t_1, t_2, and t_3), then PM_2 is the best choice for the incoming VM to minimize the total completion time. However, as shown in Fig. 1(b), the capacity of PM_2 is not sufficient. Instead of placing the incoming VM in PM_1, an alternative is to migrate one VM on PM_2 to PM_3, which makes room for the incoming VM. In that case, the incoming VM could be accepted into PM_2 to minimize the total job completion time.

However, migration has a preliminary constraint, even if the remaining capacity of PM_3 could accept the migrated VM. On the one hand, by saving space to accept this incoming VM into PM_1, we could have the minimal completion time t_2 for the incoming VM. On the other hand, since the completion time of the same VM on different PM hosts varies, for the migrated

VM, there might be a completion time increase due to the change of host. Since we aim to minimize the total completion time of the whole VMs set, if the time increase of the migrated VM is too large, we would rather select the first choice without migration. This is because migration would be meaningless in minimizing the total completion time. Therefore, we conclude that the migration constraint is that, even with the increased completion time of the migrated VM, the migration-based VM placement is still better than the direct placement in minimizing the total completion time.

Therefore, in the emulated-migration-based VM placement, we have two options, i.e., direct placement and migration-based placement. The insight behind the option selection is to compare the total job completion time of these two choices: we choose migration-based placement, if its gain of placing the incoming VM minus the switching loss of migrating the victim VM is larger than the gain of direct placement; otherwise we choose direct placement. Again, note that the migration process is only emulated in the off-line VM placement scenario. We do not actually place the VMs until all VMs' final positions are determined.

Regarding the on-line scenario where pre-information is not available, our VM placement algorithm is based on *real* VM migrations, which will inevitably introduce migration delay [3]. In fact, VM migration basically consists of transferring its memory image from the source host to the destination. Through experiments, Verma et al. [4] observed that there is a linear relationship between the active memory of a VM and the duration of migration. We refer to this delay as the *migration overhead*, since it will increase the total job completion time. Obviously, migrating a VM with larger resource demand will lead to a longer delay, thereby, a large migration overhead.

Furthermore, we conduct a study on a hybrid scheme, where we introduce a *batch* to help the VM placement in the on-line scenario. Instead of one-by-one placement, several on-line incoming VMs are reserved into a batch. Then, the information of the VMs in the batch would be known to us. In that case, we can simultaneously place them together through the off-line placement algorithm without the migration overhead. However, the batch will introduce the waiting time of the VM placement. Therefore, there exists a tradeoff between the on-line migration overhead and batch waiting time.

In this paper, we formulate the total completion time minimization VM placement problem under both off-line and on-line scenarios. Due to the NP-hardness of this problem, we propose a heuristic *migration-based VM placement* (MBVMP) algorithm to give an efficient solution. Our main contributions are summarized in the following:

- To the best of our knowledge, our work is the first one to adopt VM migration for the total completion time minimization VM placement problem, considering both off-line and on-line scenarios.
- For the off-line VM placement, we also study the case of homogeneous resource demand, and compare our heuristic algorithm with the optimal solution of maximal matching problem. The results show that the performance of our algorithm is very close to the optimal solution, verifying the high efficiency of our algorithm.
- We also conduct a study on a hybrid scheme, where a *batch* is introduced in the VM placement. In that case, we can integrate the off-line VM placement into the on-line scenario, which could avoid the migration overhead.

TABLE I
NOTATIONS

Notation	Description
VM_i	The i^{th} VM
PM_j	The j^{th} PM
R_i	The resource demand of VM_i
C_j	The remaining capacity of PM_j
x_{ij}	{0,1} variable indicates whether VM_i is placed into PM_j
t_{ij}	The job completion time of VM_i in PM_j

The remainder of the paper is organized as follows: in Section II, we study the VM placement problem under the off-line scenario, and propose our off-line MBVMP algorithm. In Section III, we perform research on the on-line scenario. Section IV introduces a hybrid scheme of VM placement. In Section V, we conduct experiments to evaluate the performance of our algorithms under both off-line and on-line scenarios. In Section VI, we introduce some previous work. Finally, we give the conclusions in Section VII.

II. OFF-LINE VM PLACEMENT

A. Off-Line Problem Description

In this subsection, we study the VM placement problem under the off-line scenario, in which we know the information about the incoming VMs set a priori (e.g., each VM's resource demand, and the total number of VM requests). Suppose a cloud system has N PMs, where the capacity of the j^{th} PM is C_j. There are M VMs waiting for the placement, and the resource demand of the i^{th} VM is R_i. Due to the heterogeneity of PMs and VMs, different VMs on the same host have various job completion times. Therefore, the element of t_{ij} in completion time matrix $[t_{ij}]$ denotes the job completion time of the i^{th} VM, which is placed into the j^{th} PM. For simplicity, we assume t_{ij} is a constant, which is not related to the current load in the j^{th} PM. Table I presents the notations for the variables. The objective is to find an optimal placement that minimizes total completion time of input jobs. Then, this optimization problem is formulated as the following:

$$Minimize \sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij}t_{ij} \tag{1}$$

$$S.T. \sum_{j=1}^{N} x_{ij} \le 1, \ \sum_{i=1}^{M} x_{ij}R_i \le C_j, \ x_{ij} \in \{0,1\} \tag{2}$$

It can be seen that this problem is essentially a multi-knapsack problem, where VMs are the items selected for the PMs (equivalent to knapsacks). Resource demands of the VMs are the weights of the items, and the job completion time is equivalent to the values of the items. Note that here we are minimizing the job completion time, rather than maximal values in the knapsack problem (but they are essentially the same). Therefore, this problem is NP-hard. Hence, we prepare to give a heuristic solution to solve it.

B. Off-Line Problem Analysis

In this section, we analyze the off-line VM placement problem, which is based on the idea of VM migration. The VMs are placed one after another. As discussed before, for VM_i, there is an optimal PM_j with minimal t_{ij}. If the remaining capacity C_j is enough, we could finish placing VM_i and switch to the next VM. Otherwise, if C_j is not sufficient, we have two options below to place it.

- Direct Placement: Among those PMs which have enough resources to accept the incoming VM, we select the one with the minimal completion time.
- (emulated) Migration-based Placement: We try to migrate one VM placed on the optimal PM to another host for saving space to accept the incoming VM. In the off-line scenario, the migration process is emulated. Therefore, we call it (emulated) Migration-based Placement.

Obviously, Direct Placement is not the optimal choice for the incoming VM, since we should not miss the opportunity of placing the incoming VM into the PM corresponding to the minimal completion time, especially under some circumstance that the completion time of the first-choice is much less than that of the others. Therefore, we want to take advantage of VM migration to do the placement.

The key for the feasibility of Migration-based Placement is to satisfy the migration constraint, i.e., the Migration-based Placement is better than Direct Placement in terms of the total completion time. To achieve our Migration-based Placement, we need to find a *qualified* victim VM placed on the optimal PM to migrate, which is not always searchable. There are three qualifications that such a qualified victim VM must meet:

1) Could save enough resources for accepting a new VM.
2) Could find a new available host with enough current capacity to accept the victim VM
3) With the increased completion increase of the migrated victim VM, the completion time by migration-based placement is less than direct placement.

If such a qualified victim VM is found on the min-completion time PM, then we would directly choose Migration-based PM. If more than one qualified victim VMs exists in the min-completion time PM, we would choose the VM with the minimal completion time increase to migrate. Of course, there could exist some special circumstances for a particular incoming VM.

- We could not find a qualified victim VM on the min-completion time PM. We then try to place the incoming VM in the next-min-completion time PM. If the incoming VM still could not be placed, we continue to try the next-next-min-completion time PM. The search process is terminated when the incoming VM is placed, or encounters the best available PM found by Direct Placement.

Algorithm 1 VM placement Algorithm

```
1: for $i$=1 to $M$ do
2:     Sort PMs increasingly by $t_{ij}$.
3:     for $j$=1 to $N$ do
4:         if $C_j$ is sufficient for $VM_i$ then
5:             Place $VM_i$ in $PM_j$
6:         if Migration-Based Placement ($VM_i$,$PM_j$) then
7:             Do VM migration and place $VM_i$ into $PM_j$
8:     if $VM_i$ is still not placed then
9:         Reject $VM_i$
```

Algorithm 2 (emulated / real) Migration-Based Placement

```
1: for all the VMs ($VM_k$) on $PM_j$ do
2:     if $VM_k$'s migration makes enough room for $VM_i$ then
3:         for all the PMs (except $PM_j$) do
4:             Find the best available PM (except $PM_j$) for $VM_k$
5:     Select the best qualified victim VM running on $PM_j$
6: if migration constraint is satisfied then
7:     return true
8: else
9:     return false
```

- After trying all the PMs, and if the incoming VM is still not placed, we have no choice but to reject it.

As we discussed before, under the the off-line scenario, VM migration is not actually implemented. However, the emulated migration process shows great promise in dynamically improving the previous placement to lower the total completion time.

C. MBVMP Algorithm

As we could observe from Algorithm 1, for each incoming VM_i, we first sort the completion time t_{ij} of each PM and accordingly rearrange the PMs set. After that, starting from the first PM (the best choice to place VM_i in minimizing total completion time) in the rearranged PMs, we try to place the incoming VM_i. If PM_j's remaining capacity C_j is sufficient, we could directly place VM_i and switch to place the next VM. Otherwise, by using the Direct Placement to test the migration constraint, we try to adopt the migration-based VM placement. Considering the two loops in Algorithm 1, the total time complexity of our off-line MBVMP algorithm is $O(M^2N)$. Again, note that the Migration-based Placement is only emulated for the off-line scenario.

In Algorithm 2, we try to achieve the Migration-based VM placement. That is, we try to find the qualified victim VM among all VMs on PM_j to migrate. For each VM_k already placed in PM_j, if it could make enough room for the incoming VM_j, we then try to find it a best new host. That is, among all the other PMs (except PM_j itself) with enough remaining capacity to accept VM_k, we select the PM that could yield the minimal completion time increase as the best new host. Among these VMs that could make enough room for the incoming VM_j, we select the best victim VM that has the minimal increased completion time. After that, we verify whether

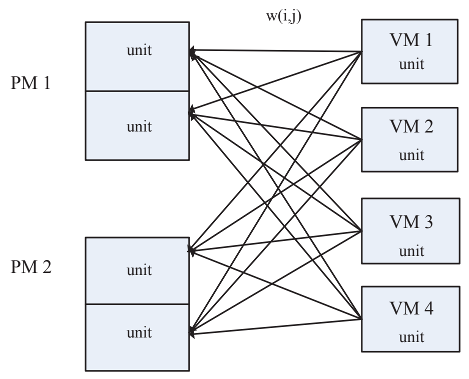


Fig. 2. Minimal Matching and VM placement

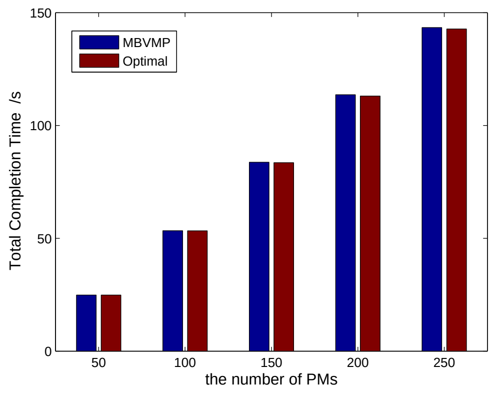


Fig. 3. Comparison results

the victim is qualified by testing the migration constraint. Through comparing the Direct Placement and Migration-based Placement in minimizing the total completion time, we could know if the selected best victim VM is qualified. If the migration constraint is satisfied, the victim VM is qualified. Then, we do the migration and place the incoming VM_i.

D. Case Study

Now we consider a special case under the off-line scenario, in which the resource type is uniform. That is, each PM's resource capacity could be sliced into unit slots, and all of the VMs have the same resource demand of one slot. Each VM is placed into the one slot in the PM, thus there exists a unique correspondence with each VM and each PM's unit slot. Given this, we could transfer the VM placement into a minimal matching problem in the Graph Theory. We would use Fig. 2 to illustrate the mapping from VM placement to minimal matching.

In Fig. 2, each VM and PM unit pair is regarded as the vertices in the bi-partite graph, and the weighted edge connecting each VM and PM unit pair refers to the completion time of placing VM in that unit of the PM. Thus, finding a minimal total completion time VM placement is equal to finding the minimal matching between VM and PM unit pairs in the weighted bi-partite graph. There is an optimal Kuhn-Munkras (KM) algorithm [5] to solve this problem, and we use an example to compare our proposed MBVMP algorithm with the optimal solution.

We assume that each PM in the cloud system has the capacity of 6 VM slots, and we vary the number of PMs from 50 to 250. We set that the total resource demands of VMs are equal to the total capacity of PMs. In that case, given the N PMs, we have a set of $N * 6$ VMs to be placed into the cloud system. Since all of the VMs have the same resource demand of one slot, each VM should have a corresponding slot in the PM to place into. We also randomly generated the set of each element in completion time matrix $[t_{ij}]$. From the comparison results in Fig. 3, we could see that the performance of our proposed algorithm is very approximate to the optimal solution. Given M incoming VMs and N PM hosts, the time complexity of the KM algorithm is $O(M^2N)$, which is equal to our off-line MBVMP algorithm.

Algorithm 3 Multiple-hop VM placement

```
1: for each PM in the set do
2:     Sort PMs set increasingly by completion time of the
       VM to be placed
3:     Select the best $PM_j$ with the minimal completion time
       for the incoming VM
4: if $C_j$ is sufficient for $VM_i$ then
5:     Place the VM in $PM_j$
6: for all the VMs running on the best PM do
7:     Select the best victim VM by comparing migration gain
8: Recursively call this algorithm to place the best victim
   VM among PM $\in P - PM_j$
```

E. Multiple-hop Migration

In the previous discussion, we only consider the one-hop migration, which will benefit for the time complexity. However, multiple-hop migration might lead to a better performance. In Algorithm 3, we present the multi-hop algorithm. Suppose the average number of placed VMs on a PM is K, then the time complexity of Algorithm 3 is K times as much as that of Algorithm 1. Obviously, Algorithm 3 introduces multiple-hop migrations with a better performance on minimizing the total completion time, as the tradeoff of a higher time complexity.

III. ON-LINE VM PLACEMENT

Under the on-line scenario, the VM requests are coming one by one without knowing the information beforehand. Hence, we place VM one after another through a emulated process. Since our MBVMP algorithm places VMs one after another for the off-line scenario, it could still work in the same way for on-line VM placement.

The major difference between on-line and off-line VM placement is that, the migration process is actually implemented. Hence, it is a real process which would introduce a migration delay, increasing the total completion time. Therefore, for the victim VM, not only do we have the completion

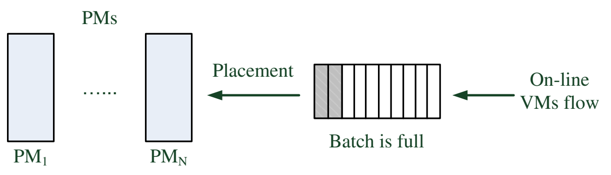


Fig. 4. Provider-oriented batch

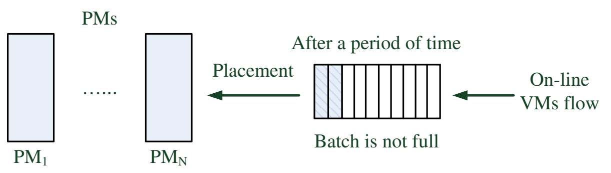


Fig. 5. User-oriented batch

time increase caused by migration, but we also have an extra migration overhead due to the on-line migration process.

Similar to the off-line scenario, we place the incoming VM one after another. When a new VM' comes, we first try to place it into the PM that corresponds to the min-completion time value. If the corresponding min-completion time PM does not have enough resources, we would have the same two options (Direct Placement and Migration-based Placement) as in the off-line scenario.

However, the migration constraint is different. Due to the migration overhead, the migration constraint would be that, even with both the migration overhead and the completion time increase, Migration-based Placement is still better than Direct Placement in minimizing the total completion time. Thus, compared to the off-line, the third item of qualifications for a qualified victim VM under on-line scenario should be changed into: Even with both the completion time increase and migration overhead, the completion time by Migration-based Placement is still better than Direct Placement. We want to choose the qualified victim VM with the minimal sum of migration overhead and completion time increase. Besides, we could have similar special circumstances as in the off-line VM placement; constrained by the literature, here we do not discuss them further more.

Besides, for on-line VM placement, we are not aware of the information of the incoming VMs, such as how many total VMs will arrive. Thus, when all the PMs are so fully-loaded that even migration is not helpful to accept more VMs, we need to stop the placement process for this PMs set. This requires us to reserve a buffer to determine when we should directly reject the new VM request. In our evaluation, we set the size of a buffer to 10. That is to say that if 10 consecutive VMs are rejected, we then conclude the VM placement.

IV. A HYBRID SCHEME

Obviously, the major disadvantage of on-line scenario is the migration delay, which will cause much overhead in the total completion time. However, the off-line scenario does not have such a drawback, because we know all the VMs' information beforehand, and they could all be placed at one time together.

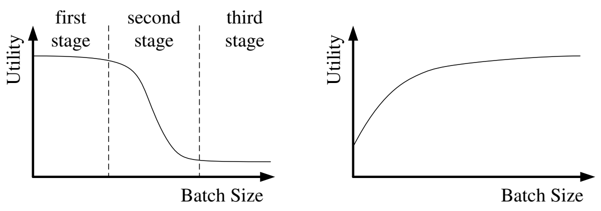


Fig. 6. The utility functions of the user (left) and the provider (right).

Thus, there is no migration overhead in the off-line scenario. This advantage of off-line scenario leads to the intuition that we could adopt a hybrid scheme that integrates the off-line scheme into the on-line scenario.

In fact, we could reserve a batch to store the on-line VM requests. In that case, the reserved VMs' information is known to us. Therefore, we could place the VMs in a batch according to the off-line scenario. In that case, we could avoid the migration overhead.

However, the batch will introduce a waiting time overhead, since we need to wait for the VMs in the batch to reach a ceratin number, and then place them together. Therefore, this waiting overhead will also increase the total completion time, although we avoid the migration overhead. Therefore, there is a tradeoff between the migration overhead and the waiting overhead. Due to this, there are two batch models: user-oriented and provider-oriented.

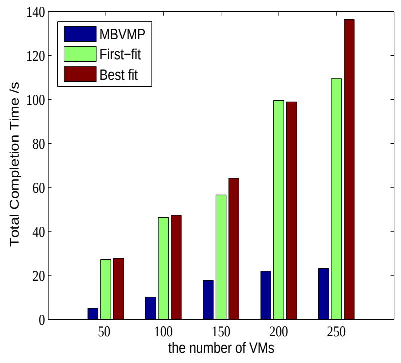
A. Provider-oriented

The provider-oriented batch model is that we place all the VMs in the batch until it is full, as shown in Fig. 4. By this, the advantage of the batch could be fully utilized, and the migration overhead could be largerly reduced, which is beneficial to the cloud providers. An extreme case is that, the batch size is very large, and we place the VMs together until the total resource demands of VMs are up to the total capacity of PMs. In that case, it is equivalent to the off-line scenario.

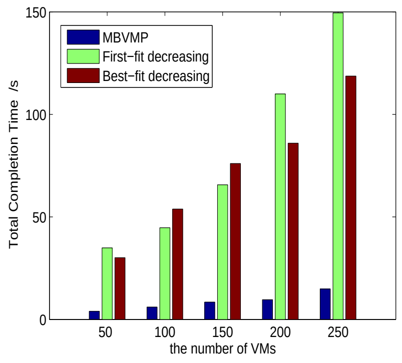
However, this waiting overhead will also largely increase the total completion time, since we have to wait for the batch to be full, and then, place the VMs together. Obviously, the larger the batch is, the larger the waiting overhead will cost. Under the on-line scenario, we do not know when a VM will arrive, and how long it will take to fulfill the batch. If the batch waiting overhead is too large, users will suffer the degradation of their applications' performance. Hence, we have another user-oriented batch model.

B. User-oriented

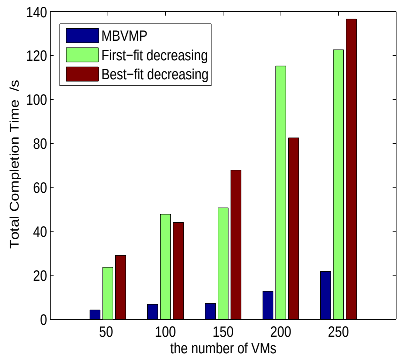
The performance of users' applications need to be guaranteed by the cloud provider, according to SLA. Therefore, the batch overhead cannot be so large as to harm the user's applications' performance requirements. The user-oriented batch model is that, after waiting for a certain period of time, we place the VMs in the batch no matter whether the batch is full or not, as shown in Fig. 5. In that case, the applications' performance could be guaranteed as the users' requirements.



(a) Number of PMs: N=100

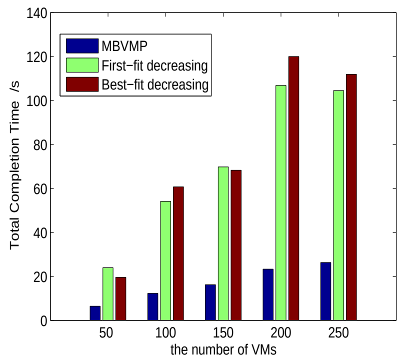


(b) Number of PMs: N=150

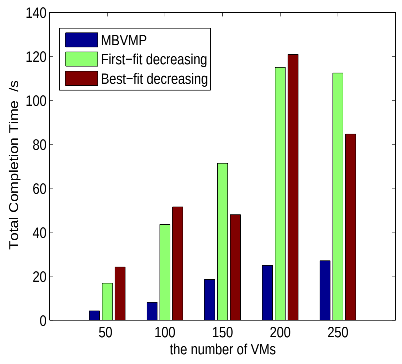


(c) Number of PMs: N=200

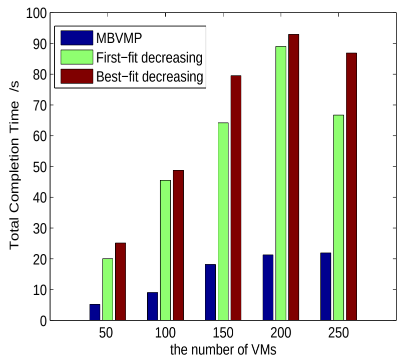
Fig. 7. Performance comparisons of total completion time vs. number of PMs



(a) variance of VMs' resource demand:[10,50]



(b) variance of VMs' resource demand:[10,100]



(c) variance of VMs' resource demand:[10,150]

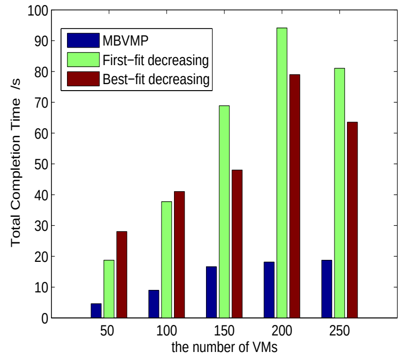
Fig. 8. Performance comparisons of total completion time vs. variance of VMs' resource demand

C. Qualitative Analysis on Batch Size

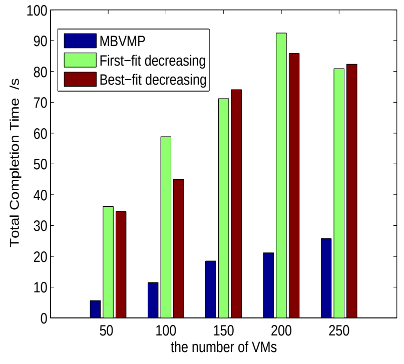
In the former sections, we have discussed the implementation method of the batch, i.e., provider-oriented batch and user-oriented batch. In this section, we give out qualitative analysis on the batch size. According to [6], the time consumption of virtual machine migrations usually varies from 30 to 150 seconds, due to the transmission time of the memory (usually several GBs). Therefore, the bandwidth has significant influence on the migration time. On the other hand, the acceptable waiting time of the user greatly depends on the application. For example, web services [7] generally respond in less than 1 second. In this case, migration-based placement is not practical, i.e., the batch is useless and we should place the VMs as soon as possible. Therefore, the batch model should be applied to user applications which takes a relatively long time (e.g., several hours or more).

For most cases, we can determine the batch size through a utility model. The user's utility monotonously decreases with respect to the batch size, while the providers's utility monotonously increases with respect to the batch size. The objective is to maximize the summation of the user's utility and the providers's utility. Moreover, the utility function should exhibit some certain properties, as shown in Fig. 6. According to the experience of daily life, the utility function of the user should have three stages as follows. (1) In the first stage, the utility is high and decreases slowly, since the user is not sensitive to a small waiting time. For example, the waiting time of 2 and 10 milliseconds do not make too much difference, since people is not aware of such a small time duration. (2) Generally speaking, the user has an expected maximal waiting time (no one wants to wait for a too long time). Once this time is approaching, the utility decreases quickly in the second stage. (3) In the third stage, the user has been waiting for a too long time. The utility is very low such that waiting for an addition time does not greatly turn down the utility. Meanwhile, the utility function of the provider is very simple, since it should has a *diminishing return* property. The utility gain should decrease with respect to the batch size.

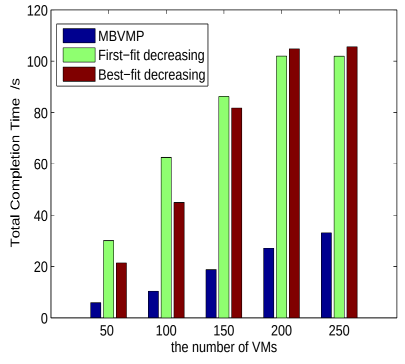
The two utility functions should be designed according to real systems and applications, i.e, different systems and applications should have different function format, although they should have the properties discussed above. If the user's utility dominates (usually in some real-time applications and user interactive applications), then a small batch size is desired. If the provider's utility dominates (usually in some large computation applications), then a large batch size is desired. This tradeoff always exists in the cloud system.



(a) variance of PMs' capacity:[50,150]



(b) variance of PMs' capacity:[100,150]



(c) variance of PMs' capacity:[150,150]

Fig. 9. Performance comparisons of total completion time vs. variance of PMs' capacity

V. Performance Evaluation

In this section, we evaluate our proposed MBVMP algorithms under both off-line and on-line scenarios. We made comparisons with first-fit and best-fit algorithms. Notably, for the off-line scenario, we adopt the first-fit-decreasing and best-fit-decreasing algorithms [8]. In order to compare the performance of one-hop and multiple-hop migration, we conduct extended simulation by adopting two-hop migration under off-line scenarios. As for the on-line, multiple-hop migration would largely increase the migration cost. Therefore, we usually do not consider it as an alternative in the on-line VM placement.

A. Off-Line MBVMP Algorithm Evaluation

We evaluate the performance of our off-line MBVMP algorithm under three groups of simulations on the total completion time. We use mathematical abstraction to describe the physical meaning of PM's capacity and VM's resource demand. The measuring unit of resources is unit slot, which could be easily interpreted to real configuration. For instance, one unit slot for Amazon S3 could be interpreted as unit storage space for renting. The set of each element in completion time matrix $[t_{ij}]$ is randomly generated.

1) Simulation Settings:

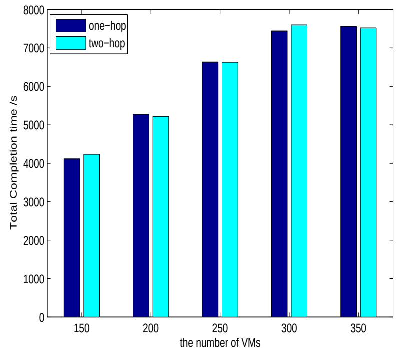
- Group 1: We deploy PMs' numbers N=100,150 and 200. We set all PMs' resource capacity range [150,150] and all the VMs' resource demand range [10,100], which conforms to the real proportional relationship.
- Group 2: We deploy all VMs' resource demand ranges [10,50], [10,100], [10,150]. We set PMs' number N=150 and all PMs' current capacity range [150,150].
- Group 3: We deploy all PMs' current capacity ranges [50,150], [100,150], [150,150]. We set PMs' number N=100 and all VMs' resource demand range [10,100].

2) Simulations Results: Results for three groups of simulations are shown in Figs. 7, 8 and 9. From Figs. 7, 8 and 9, we can see that under all settings of the number of PMs, our MBVMP algorithm could achieve a significantly higher total completion time than the other two heuristic algorithms. Besides that, we could still have the following observations:

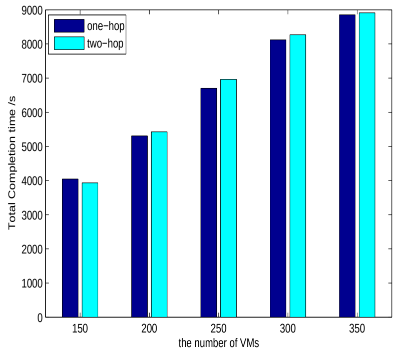
1) In each of subfigures (a), (b) and (c), with the increase of the number of VMs, the total completion time grows. This is quite straightforward, since more VMs are placed in the PMs, thus bringing more completion time.
2) By comparison of (a), (b) and (c) of Fig. 7, we find that when the number of PMs increases, the gap of the total completion time between the 250 VMs and 200 VMs is increasing. This is due to the fact that, when PMs numbers are few, given a 250 VMs input, many VMs actually cannot be accepted. Thus, there is not much difference between 200 VMs input and 250 VMs. However, due to the increase of PM hosts, and thereby more resources, more VMs could be accepted into the cloud system, and we could get an increase in total completion time. This reason also applies to comparisons between Figs. 9 (a), (b) and (c), due to the increase of PM's capacity. On the other hand, Figs. 8 (a), (b) and (c) have the opposite trend, due to the increase of VM's resource demand.

B. On-Line MBVMP Algorithm Evaluation

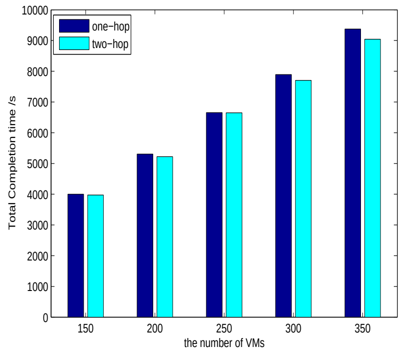
For the on-line scenario, we could not know how many VM requests would come, therefore, we set it so that if 10 consecutive VM requests could not be placed, we then would stop the whole VM placement process. As discussed before, migration overhead has a positive correlation with the VM's resource demand. For analysis simplicity without loss of generality, we set a linear coefficient α to show that the migration overhead is α times the VM_i's resource demand, which is c_i=$\alpha \times R_i$. We also conducted three groups of simulations with settings similar to the off-line scenario. In the first group, we change the number of PMs, while keeping the PMs' capacity and migration cost coefficient stable. In the second group, we vary the migration cost coefficient α to test its effect on the performance. At last, we vary the variance of PMs' capacities, while keeping the number of PMs and cost coefficient α stable. Simulation results are presented in Table II. Table II shows that our MBVMP algorithm outperforms



(a) Number of PMs: N=100

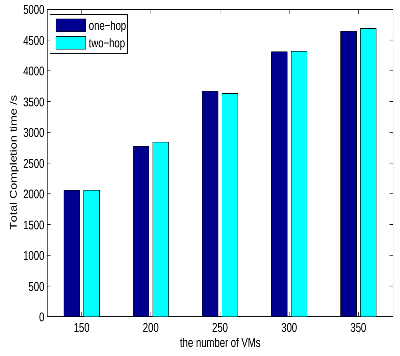


(b) Number of PMs: N=150

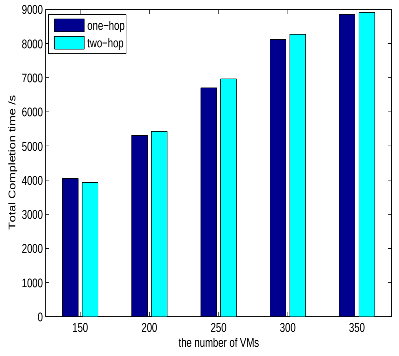


(c) Number of PMs: N=200

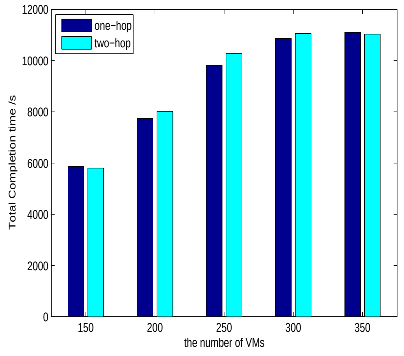
Fig. 10. Performance comparisons of total completion time vs. number of PMs



(a) variance of VMs' resource demand:[10,50]



(b) variance of VMs' resource demand:[10,100]



(c) variance of VMs' resource demand:[10,150]

Fig. 11. Performance comparisons of total completion time vs. variance of VMs' resource demand

the best-fit and first-fit algorithms under the on-line scenario in all settings of variables.

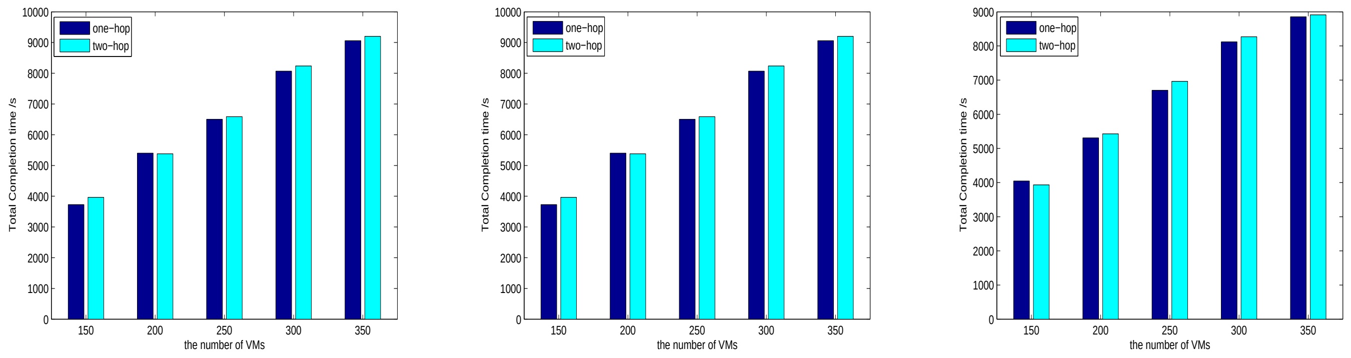
C. One-hop and two-hop comparison

We can see from the comparsion in Figs. 10, 11, and 12, the performance is almost comparable between one-hop and two-hop migration. This indicates that two-hop migration will not largely increase the performance, however, with a longer time complexity. Therefore, our one-hop migration-based virtual machine placement algorithm will be good enough.

VI. Related Work

As a new paradigm of distributed computing, cloud computing has brought several key advantages into our lives, such as on-demand scaling and pay-as-you-go metered service. The development of virtualization technologies has boosted the spread of cloud computing. Through management by virtual machine monitor (VMM) [9–12], the physical resources of one PM could be sliced into multiple VMs. Such resource multiplexing largely reduces the total cost of ownership, and significantly improves the resource utilization. As a contribution of virtualization technology, VM migraton [13–18] improves the resource rearrangement on the fly.

Much work has been done regarding VM placement in the cloud computing environment, which is a complicated task involving various constraints, including performance [19], availability [20], network [21], and cost [22]. Economic interests are one popular topic that shows up in research literature [23, 24]. They tried to find an optimal VM placement that could either minimize the revenue for the cloud provider, or minimize the costs for the customers. In [24], for minimizing the total economic completion time of the cloud provider, the author introduced an SLA-based dynamic resource allocation. The pricing mechanisms are related to the performance of QoS that the cloud provider could guarantee. The better performance the cloud provider could offer, the more revenue the cloud provider could obtain. Since different service requests have different pricing, higher priced service can get more resource provisioning from the cloud provider. The author also used a convex optimization to present the optimal resource allocation. On the contrary, the author in [23] proposed an optimal VM placement with the objective of minimizing the total renting of the users. In this paper, the author gives another pricing mechanism, referring to two payment plans: reservation plan and on-demand plan. Since the total resource demand is uncertain, and reservation plan has to be decided in advance, it may not meet the future demands of the user. For that reason, the author used the optimal solution of stochastic integer programming to give an optimal VM placement that



(a) variance of PMs' capacity:[50,150] (b) variance of PMs' capacity:[100,150] (c) variance of PMs' capacity:[150,150]

Fig. 12. Performance comparisons of total completion time vs. variance of PMs' capacity

TABLE II
PERFORMANCE COMPARISON FOR ON-LINE SCENARIO

Total Completion Time /s	Number of PMs			Coefficient of Migration Cost			Variance of PMs' Capacity		
	100	150	200	0.1	1	10	[10,50]	[10,100]	[10,150]
MBVMP	52.4475	55.2926	83.9606	56.1748	58.3576	53.1326	51.2983	53.9628	52.8962
First-fit	140.7237	214.7462	265.3581	150.1385	254.9847	284.9215	147.4251	221.2317	251.1223
Best-fit	130.2293	206.0019	233.3534	127.9724	215.9011	201.1946	125.9402	237.7812	263.9472

could minimize the total renting. However, the VM placement problem in [23] and [24] could achieve an optimal solution, which is not a common case. Many VM placement issues are NP-hard, thus we need to find a good heuristic algorithm to solve the problem, such as the first-fit and best-fit greedy algorithm used in [8].

Apart from the problems above aiming to get an optimal economic interest, network is another constraint needing consideration in the VM placement problem. In [21], a network-aware VM placement is proposed. In [21], when performing VM placement, not only the physical resources (like CPU and memory) are considered, but also the traffic demands between different VMs are taken into account. The author gave a heuristic algorithm to allocate placement to satisfy both the communication demands and physical resource restrictions. In [25], Oktopus uses the hose model to abstract the tenant's bandwidth request, including both virtual cluster and oversubscribed virtual clusters. The virtual cluster provides tenants with guarantees on the network bandwidth they demand, which, according to [26], could be interpreted as min-guarantee requirements. In [27], the author proposes to minimize the traffic cost through VM placement. Their objective is to place VMs that have large communication requirements close to each other, so as to reduce network capacity needs in the datacenter. A quadratic-assignment formulation of the traffic-aware placement problem is presented and solved with an approximation algorithm. However, their algorithm did not take into account the VM migration traffic, leading to a near complete shuffling of almost all VMs in each round. To alleviate this, VirtualKnotter [28] minimizes the continuous congestion mainly in core and aggregation links with controllable migration traffic, which enables online VM replacement. In [29], a distributed cloud system is studied. The authors propose a network-aware VM placement algorithm, which consider the difference of latencies between inter-data-center and inner-data-center. They developed data-center selection algorithms for VM placement that minimize the maximum distance between the selected data centers.

Moreover, our work is also potentially related to the process assignment problem in both symmetric and asymmetric multi-processing (SMP and AMP [30]). In that technology, processes can also be migrated from one processor core to another. In the case of non-uniform memory access (NUMA) model, the pages on the NUMA nodes are migrated. The idea of process migration is further applied into our VM placement strategy, where we emulate a VM migration process for the objective of off-line VM placement.

VII. CONCLUSION

In this paper, we studied the VM placement problem, focusing on minimizing the total completion time of cloud providers under both off-line and on-line scenarios. Due to the NP-hardness of this problem, we propose our heuristic migration-based VM placement algorithm. In particular, for the off-line scenario and homogeneous resource type, we compare our algorithm with the optimal maximal matching solution, which shows a high approximation of our algorithm with the optimal one. Furthermore, we study the hybrid scheme of integrating off-line placement into on-line scenario, and propose two batch models considering users or providers separately. The simulation shows the high efficiency of our proposed heuristic algorithms, compared to the best-fit and first-fit heuristic algorithms.

Acknowledgment

This research was supported in part by NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167.

References

[1] W. Wang, Y. Zhang, B. Lin, X. Wu, and K. Miao, "Secured and reliable vm migration in personal cloud," in *Proceedings of ICCET 2010*, pp. 705–709.

[2] H. Xu and B. Li, "Egalitarian stable matching for vm migration in cloud computing," in *Proceedings of IEEE INFOCOM WKSHPS 2011*, pp. 631–636.

[3] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proceedings of CloudCom 2009*, pp. 254–265.

[4] A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Proceedings of Middleware 2010*, pp. 11–16.

[5] P. Tichavsky and Z. Koldovsky, "Optimal pairing of signal components separated by blind techniques," *IEEE Signal Processing Letters*, no. 2, pp. 119–122, 2004.

[6] T. Wood, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *ACM SIGPLAN Notices*, vol. 46, pp. 121–132, 2011.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Proceedings of GECCO 2005*, pp. 1069–1075.

[8] J. Xu and J. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proceedings of CPSCom 2010*, pp. 179–188.

[9] J. R. Lange and P. Dinda, "Symcall: symbiotic virtualization through vmm-to-guest upcalls," in *Proceedings of ACM VEE 2011*, pp. 193–204.

[10] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Vmm-based hidden process detection and identification using lycosid," in *Proceedings of ACM VEE 2008*, pp. 91–100.

[11] M. Xu, X. Jiang, R. Sandhu, and X. Zhang, "Towards a vmm-based usage control framework for os kernel integrity protection," in *Proceedings of ACM SACMAT 2007*, pp. 71–80.

[12] A. Ranadive, A. Gavrilovska, and K. Schwan, "Ibmon: monitoring vmm-bypass capable infiniband devices using memory introspection," in *Proceedings of HPCVirt 2009*, pp. 25–32.

[13] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, "Enabling secure vm-vtpm migration in private clouds," in *ACSAC 2011*, pp. 187–196.

[14] S. Kumar and K. Schwan, "Netchannel: a vmm-level mechanism for continuous, transparentdevice access during vm migration," in *Proceedings of ACM VEE 2008*, pp. 31–40.

[15] N. Jain, I. Menache, J. S. Naor, and F. B. Shepherd, "Topology-aware vm migration in bandwidth oversubscribed datacenter networks," in *Proceedings of ICALP 2012*, pp. 586–597.

[16] A. Surie, H. A. Lagar-Cavilla, E. de Lara, and M. Satyanarayanan, "Low-bandwidth vm migration via opportunistic replay," in *Proceedings of HotMobile 2008*, pp. 74–79.

[17] H. W. Choi, H. Kwak, A. Sohn, and K. Chung, "Autonomous learning for efficient resource utilization of dynamic vm migration," in *Proceedings of ICS 2008*, pp. 185–194.

[18] K. Srinivasan, S. Yuuw, and T. J. Adelmeyer, "Dynamic vm migration: assessing its risks &#38; rewards using a benchmark," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 317–322, 2011.

[19] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Proceedings of ICAC 2008*, pp. 3–12.

[20] E. Bin, O. Biran, O. Boni, E. Hadad, E. Kolodner, Y. Moatti, and D. Lorenz, "Guaranteeing high availability goals for virtual machine placement," in *Proceedings of IEEE ICDCS 2011*, pp. 700–709.

[21] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Proceedings of IEEE GCC 2010*, pp. 87–92.

[22] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *Proceedings of IEEE ICDCS 2011*, pp. 559–570.

[23] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of IEEE APSCC 2009*, pp. 103–110.

[24] G. Feng, S. Garg, R. Buyya, and W. Li, "Revenue maximization using adaptive resource provisioning in cloud computing environments," in *Proceedings of GRID 2012*, pp. 192–200.

[25] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of ACM SIGCOMM 2011*, pp. 242–253.

[26] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proceedings of ACM HotNets-X 2011*, pp. 22:1–22:6.

[27] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM 2010*, pp. 1–9, 2010.

[28] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu, "Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter," in *Proceedings of IEEE ICDCS 2012*, pp. 12–21.

[29] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in *Proceedings of IEEE INFOCOM 2012*, pp. 963–971, 2012.

[30] Y. Jiang, X. Shen, J. Chen, and R. Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors," in *Proceedings of ACM PACT 2008*, pp. 220–229.