Migration-based Virtual Machine Placement in Cloud Systems

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

***Abstract*—Cloud computing is an emerging technology that greatly shapes our lives, where users run jobs on virtual machines (VMs) on physical machines (PMs) provided by a cloud provider, saving the investment in upfront infrastructures. Due to the heterogeneity of various jobs, different VMs on the same PM can have different job completion times. Meanwhile, the PMs are also heterogeneous. Therefore, different VM placements have different job completion times. 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 emulated VM migration, while the on-line VM placement is solved by a real VM migration process. The migration algorithm is a heuristic approach, where we place the VM to its best PM directly, as long as it has enough capacity. Otherwise, if the migration constraint is satisfied, we migrate another VM from this PM to accommodate the new VM. Furthermore, we study a hybrid scheme where a batch is employed to accept upcoming VMs for the on-line scenario. Evaluation results prove the high efficiency of our algorithms.**

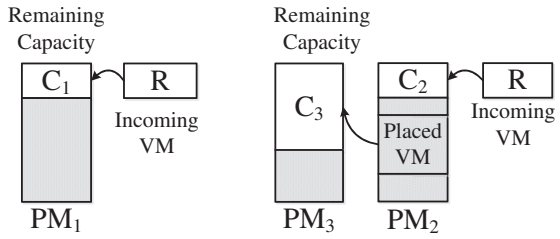
***Index Terms*—VM migration, job completion time, off-line, on-line, VM placement.**

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 can have different job completion times. Moreover, in most circumstances, the host PMs are also heterogeneous, such as different CPU architectures, different OSes, differing amounts of memory, storage, etc. Hence, the same VM placed on different PMs will 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 instance, when all PMs are heavily loaded without enough capacity to place the new VM, through migrating one running VM to another host, we can save space to accept the incoming VM. In this paper, we consider both off-line and on-line VM placement problems on minimizing the total job completion time.



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

Fig. 1. An illustration of the VM placement

Under the off-line scenario, the information of input VMs are a known priori. Therefore, we propose a VM placement algorithm through a heuristic *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 VMs 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 can 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. Then, to minimize the total completion time, PM_2 is the best choice for the incoming VM. 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 method is to migrate one VM on PM_2 to PM_3, which makes room for the incoming VM. In that case, the incoming VM can be accepted into PM_2 to minimize the total job completion time.

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

978-1-4799-0568-3/13/$31.00

time increase due to the change of host. Since we aim to minimize the total completion time of the whole set of VMs, if the time increase of the migrated VM is too large, we would rather select the first choice without migration. This is because migration will 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 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 will 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 increase 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 can be outlined by 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 this case, we integrate the off-line VM placement into the on-line scenario, which can avoid the migration overhead.

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, such as 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 denoted as 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 not related to the current load in the j^{th} PM, i.e., t_{ij} is a constant. Table I presents the notations for the variables. We need to find an optimal placement in order to gain the minimal total completion time of input jobs. The objective function can be expressed mathematically as:

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

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

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

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

VM_i, there is an optimal PM_j with minimal t_{ij}. If the remaining capacity C_j is enough, we can finish placing VM_i and switch to the next VM. Otherwise, if C_j is not sufficient, we have two options 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 to make 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, which is that Migration-based Placement is better than Direct Placement in minimizing 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) Can save enough resources for accepting a new VM.
2) Can 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 can 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.
- After trying all the PMs, and if the incoming VM is still not placed, we have no choice but to reject it.

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
```

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 can 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 can 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. For the off-line scenario, it is an emulated Migration-based Placement.

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 can 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 can yield the minimal completion time increase as the best new host. Among these VMs that can 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 the victim is qualified by testing the migration constraint.

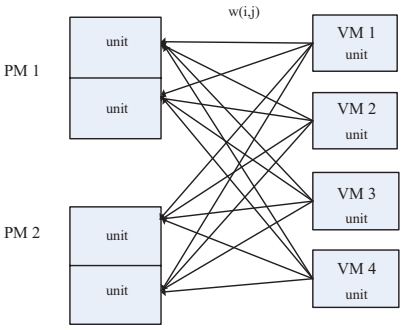


Fig. 2. Minimal Matching and VM placement

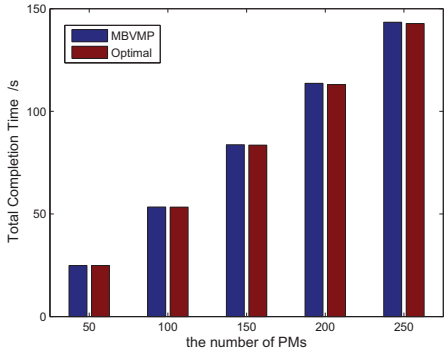


Fig. 3. Comparison results

Through comparing the Direct Placement and Migration-based Placement in minimizing the total completion time, we can 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.

With M VMs to be placed, assuming the average number of placed VMs on a PM is K, then, the time complexity of Algorithm 3 is $O(KN)$, which is no more than $O(M)$. Considering the two loops in algorithm 1, the total time complexity of our off-line MBVMP algorithm is $O(M^2N)$.

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 can 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 can transfer the VM placement into a minimal matching problem in the Graph Theory. We will 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 a 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 be placed into. We also randomly generated the set of each element in completion time matrix $[t_{ij}]$. From the comparison results in Fig. 3, we can 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.

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 can only place VMs one after another. Since our MBVMP algorithm places VMs one after another for the off-line scenario, it can 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 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 VMs 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 will 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 will be that, even with both the migration overhead and the increased completion time, 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 increased completion time. Besides, we can have similar special circumstances as in the off-line

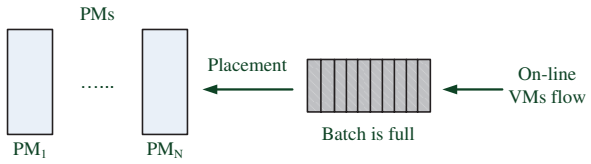


Fig. 4. Provider-oriented batch

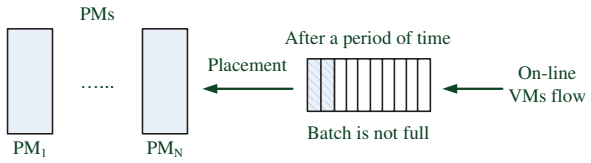


Fig. 5. User-oriented batch

VM placement; constrained by the literature, we do not repeat them here.

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 set of PMs. 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 can all be placed at one time together. Thus, there is no migration overhead in the off-line scenario. This advantage of off-line scenario leads to the intuition that we can adopt a hybrid scheme that integrates the off-line scheme into the on-line scenario.

In fact, we can reserve a batch to store the on-line VM requests. In that case, the reserved VMs' information is known to us. Therefore, we can place the VMs in a batch according to the off-line scenario. In that case, we can 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 can be fully utilized, and the migration overhead can 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 the 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 can be guaranteed, favoring the requirement of users.

V. PERFORMANCE EVALUATION

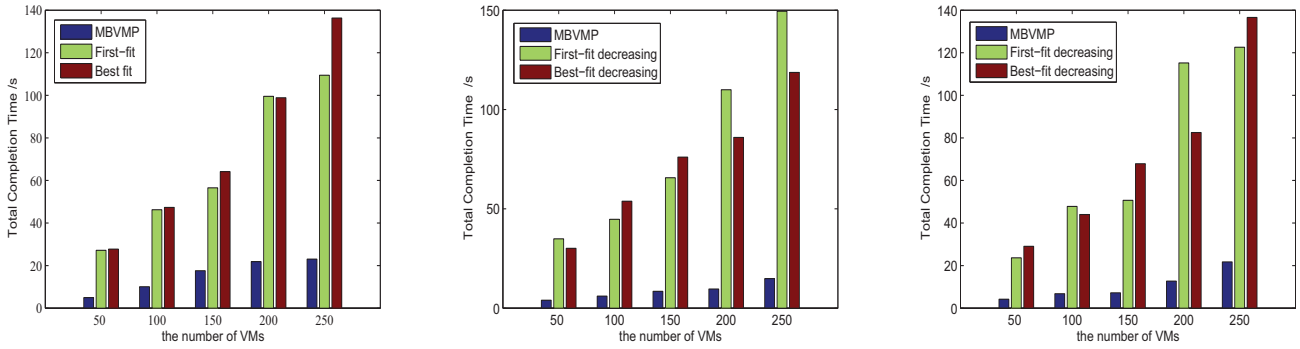
In this section, we evaluate our proposed MBVMP algorithms under both off-line and on-line scenarios. We made comparisons with first-fit and best-fit algorithms. Notably, for the off-line scenario, we adopt the first-fit-decreasing and best-fit-decreasing algorithms [6].

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 can be easily interpreted to real configuration. For instance, one unit slot for Amazon S3 can be interpreted as unit storage space for renting. The set of each element in completion time matrix $[t_{ij}]$ is randomly generated.

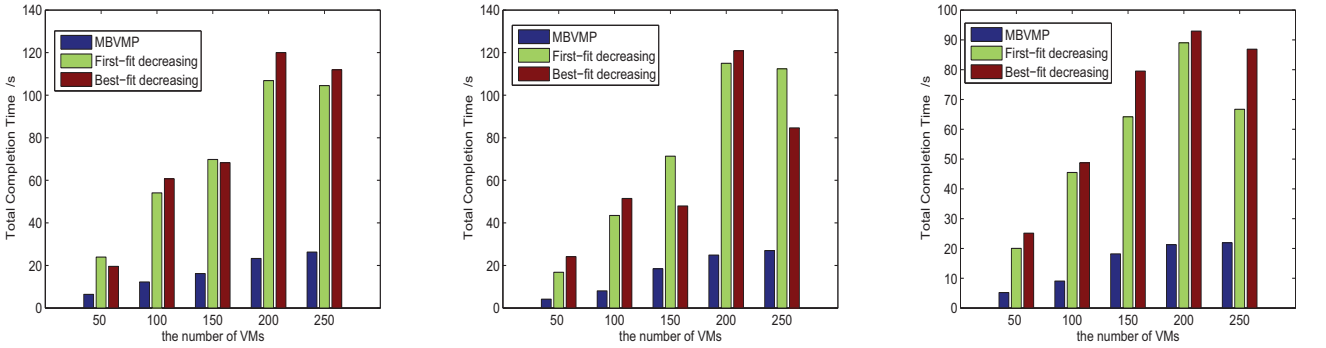
1) Simulation Settings:

- 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].



(a) Number of PMs: N=100 (b) Number of PMs: N=150 (c) Number of PMs: N=200

Fig. 6. 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. 7. Performance comparisons of total completion time vs. variance of VMs' resource demand

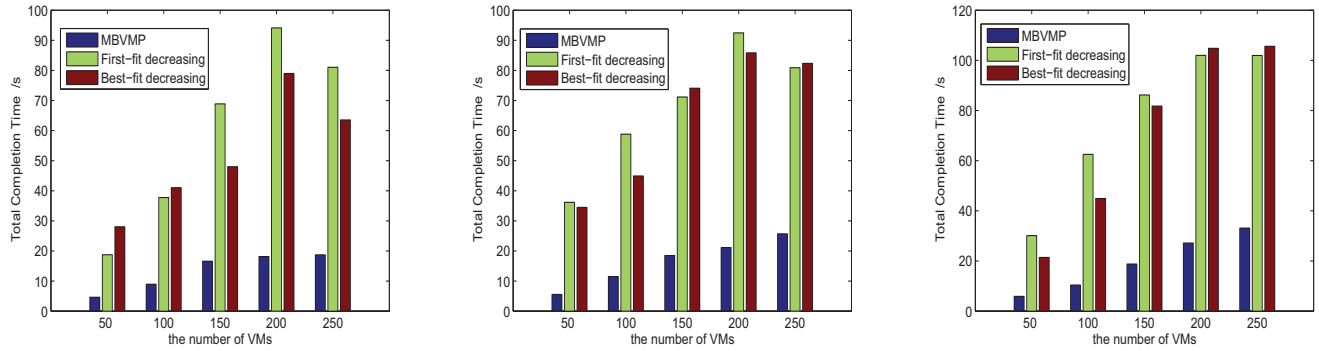
2) Simulations Results:

Results for three groups of simulations are shown in Figs. 6, 7 and 8. From Figs. 6, 7 and 8, we can see that under all settings of the number of PMs, our MBVMP algorithm can achieve a significantly higher total completion time than the other two heuristic algorithms. Besides that, we can still have the following observations:

1) In each of subfigures (a), (b) and (c), with the increase in 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. 6, we find that when the number of PMs increases, the gap of the total completion time between the 250 VMs and 200 VMs increases. 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 can be accepted into the cloud system, and we can get an increase in total completion time. This reason also applies to comparisons between Figs. 8 (a), (b) and (c), due to the increase of PM's capacity. On the other hand, Figs. 7 (a), (b) and (c) have the opposite trend, due to the increase of the VM's resource demand.

B. On-Line MBVMP Algorithm Evaluation

For the on-line scenario, we can not know how many VM requests will come, therefore, we set it so that if 10 consecutive VM requests can not be placed, we then will stop the whole VM placement process. As discussed before, migration overhead has a positive correlation with the VM's resource demand. For analysis simplicity and 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 the best-fit and first-fit algorithms under the on-line scenario in all settings of variables.



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

Fig. 8. Performance comparisons of total completion time vs. variance of PMs' capacity

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) [7–10], the physical resources of one PM can 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 [11–16] 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 [17], availability [18], network [19], and cost [20]. Economic interests are one popular topic that shows up in research literature [21, 22]. 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 [22], 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 can offer, the more revenue the cloud provider can 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 [21] 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 can minimize the total renting. However, the VM placement problem in [21] and [22] can 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 [6].

Apart from the problems above aiming to get an optimal economic interest, network is another constraint needing consideration in the VM placement problem. In [19], a network-aware VM placement is proposed. In [19], 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 [23], 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 [24], can be interpreted as min-guarantee requirements. In [25], 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 [26] minimizes the continuous congestion mainly in core and aggregation links with controllable migration traffic, which enables online VM replacement. In [27], a distributed cloud system is studied. The authors propose a network-aware VM placement algorithm in distributed cloud systems, 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.

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

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 proposed 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*, vol. 1, pp. V1–705 –V1–709, april 2010.

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

[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 09*, (Berlin, Heidelberg), pp. 254–265, Springer-Verlag, 2009.

[4] A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Proceedings Middleware Industrial Track '10*, (New York, NY, USA), pp. 11–16, ACM, 2010.

[5] P. Tichavsky and Z. Koldovsky, "Optimal pairing of signal components separated by blind techniques," *IEEE Signal Processing Letters*, pp. 119 – 122, feb. 2004.

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

[7] J. R. Lange and P. Dinda, "Symcall: symbiotic virtualization through vmm-to-guest upcalls," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '11, (New York, NY, USA), pp. 193–204, ACM, 2011.

[8] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Vmm-based hidden process detection and identification using lycosid," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, (New York, NY, USA), pp. 91–100, ACM, 2008.

[9] M. Xu, X. Jiang, R. Sandhu, and X. Zhang, "Towards a vmm-based usage control framework for os kernel integrity protection," in *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, (New York, NY, USA), pp. 71–80, ACM, 2007.

[10] A. Ranadive, A. Gavrilovska, and K. Schwan, "Ibmon: monitoring vmm-bypass capable infiniband devices using memory introspection," in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, HPCVirt '09, (New York, NY, USA), pp. 25–32, ACM, 2009.

[11] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun, "Enabling secure vm-vtpm migration in private clouds," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, (New York, NY, USA), pp. 187–196, ACM, 2011.

[12] S. Kumar and K. Schwan, "Netchannel: a vmm-level mechanism for continuous, transparentdevice access during vm migration," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, (New York, NY, USA), pp. 31–40, ACM, 2008.

[13] N. Jain, I. Menache, J. S. Naor, and F. B. Shepherd, "Topology-aware vm migration in bandwidth oversubscribed datacenter networks," in *Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, (Berlin, Heidelberg), pp. 586–597, Springer-Verlag, 2012.

[14] A. Surie, H. A. Lagar-Cavilla, E. de Lara, and M. Satyanarayanan, "Low-bandwidth vm migration via opportunistic replay," in *Proceedings of the 9th workshop on Mobile computing systems and applications*, HotMobile '08, (New York, NY, USA), pp. 74–79, ACM, 2008.

[15] H. W. Choi, H. Kwak, A. Sohn, and K. Chung, "Autonomous learning for efficient resource utilization of dynamic vm migration," in *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, (New York, NY, USA), pp. 185–194, ACM, 2008.

[16] K. Srinivasan, S. Yuuw, and T. J. Adelmeyer, "Dynamic vm migration: assessing its risks & rewards using a benchmark," *SIGSOFT Softw. Eng. Notes*, vol. 36, pp. 317–322, Sept. 2011.

[17] 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, june 2008.

[18] 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 ICDCS 2011*, pp. 700 –709, june 2011.

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

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

[21] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of IEEE Asia-Pacific Services Computing Conference, 2009*, pp. 103 –110, dec. 2009.

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

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

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

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

[26] 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 *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pp. 12–21, 2012.

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