VSCM: a Virtual Server Consolidation Manager for Cluster

Yang Liu[1], Yunchu Liu[1], Hanli Bai[1]

[1] Computational Aerodynamics Institute
China Aerodynamics Research & Development Center,
621000 Mianyang, China
cai@cardc.cn

Abstract. Virtual server consolidation is to use virtual machines to encapsulate applications which are running on multiple physical servers in the cluster and then integrate them into a small number of servers. Nowadays, with the expanding of enterprise-class data centers, virtual server consolidation can reduce large number of servers to help the enterprises reduce hardware and operating costs significantly and improve server utilization greatly. In this paper, we propose the VSCM manager for virtual cluster, which solves the problems in the consolidation from a globally optimal view and also takes migration overhead into account. Experiment results in virtual cluster demonstrate that, VSCM can greatly reduce the number of servers and the migration overhead.

Keywords: Virtual Server Consolidation, Virtual Cluster, Virtual Machine Bin Packing, Virtual Machine Migration.

1 Introduction

With the development of information technology, to meet the growing business, many companies start to build their own data centers. In the twenty-first century, enterprise-class data centers have been expanding, in which the growth rate of the number of servers is very impressive. However, the high costs of hardware and operation and the low utilization of these servers become very important problems. Virtual Server Consolidation (VSC), which can greatly reduce the number of servers, becomes a widespread-used technology for enterprise-class data centers to reduce hardware and operating costs and improve the server utilization. To use virtual machines (VM) to encapsulate applications which are running on multiple physical servers (PS) in the cluster and then integrate them into a small number of servers, VSC can reduce the number of servers greatly and improve their utilization significantly.

Several approaches [1-2] to VSC have been proposed. However, these approaches use heuristics from a locally optimal perspective according to some strategies, in which resource demands of VMs and resource residuals of PSs could not be considered from a globally optimal perspective. As a result, missing many chances to reduce the number of servers, these approaches could hardly achieve the minimum number of servers. Furthermore, they focus on how to make a new placement in

which a VM is able to migrate to another PS, ignoring the migration overhead. Making use of constraint programming (CP) [3], Hermenier et al. [4] have solved above problems. But, they make this accomplishment in a homogeneous cluster environment, and one PS is allowed to run only one VM at a time which is much different from the real cluster of data center. In the real environment, using CP will produce significant time overhead that data center can not afford.

In this paper, we propose a new approach to VSC in virtual cluster that consider both the problem of allocating the VMs to available PSs and the problem of how to migrate the VMs to these servers. Our consolidation manager Virtual Server Consolidation Manager (VSCM) works in two phases. In the first phase, according to the measurements of CPU and memory usages of VMs and servers, VSCM uses Globally Optimal Virtual Machine Bin Packing (GOVMBP) algorithm to calculate a VM placement from the point of globally optimal view, with the objective of reducing the number of servers as much as possible. And in the second phase, based on the feedback mechanism, VSCM uses Feedback based Virtual Machine Migration (FVMM) algorithm to improve the placement and remove the constraints on migration. In our experiment, we simulate such a cluster environment in which there are 64 PSs and 100 VMs that are randomly generated. Compared to consolidation based on previously-used First Fit Decreasing (FFD), GOVMBP saves 10.22% of PSs in average. And FVMM gains a saving of 67.6% of migration step and 34.7% of migration overhead. Promising experiment results show that VSCM not only greatly reduces the number of servers but also significantly reduces the migration overhead.

The rest of the paper is organized as follows. We state the problems in the two phases of VSCM in Section 2. In Section 3, we describe our Virtual Server Consolidation Manager in details. Results from our experimental evaluation are reported in Section 4. We compare our work with related research efforts in Section 5 and conclude in Section 6.

2 Problem Statements

As mentioned above, VSCM works in two phases. The first phase calculates the minimum number *n* of servers that are necessary to hold all the VMs. We refer to this problem as the Virtual Machine Bin Packing Problem (VMBPP). The second phase reduces the migration overhead as much as possible, giving the number *n*. We refer to this problem as the Virtual Machine Migration Problem (VMMP).

2.1 The Virtual Machine Bin Packing Problem

The goal of the VMBPP is to calculate the minimum number *n* of servers that are necessary to hold all the VMs, according to the measurements of CPU and memory usages of VMs and servers. Consider the servers as bins, their resource capacity as bins' volume, VMs as items, and VMs' resource demands as items' volume, this problem can be expressed as Bin Packing problem [5], a classic NP-hard problem.

A VM cannot be allocated to a PS until the server has enough resource, for example, CPU, memory and etc. Taking into account that we do not study the

quantity of VM allocated to one CPU core, for simple, we prescribe that one Virtual CPU (VCPU) of VM occupy one CPU core. Considering these two factors, CPU and memory, VMBPP can be reduced to 2-D Bin Packing problem [6].

Used by previous approaches [1, 7-8] as the solution to VMBPP, FFD works as follows: firstly, based on latest resource measurements of VMs and PSs, ranks the VMs in the decreasing order of resource demands; then, for each VM, if the residual resource of PS is enough for the VM, the VM will stay, if not, migrate this VM to the first PS which can hold it; if there's no active PS that can be migrated to, activate a new PS to hold the VM. The description of FFD discloses a strict restriction: VM can not migrate to a PS that has not enough residual resource. Although ensuring the feasibility of the VM migration, this restriction makes FFD a locally optimal solution which misses many opportunities to reduce the number *n*.

Therefore, FFD can not be competent for VMBPP, and only a solution standing from a globally optimal view is able to dig out the relationship between the demands of VMs and residuals of PSs, to make a plan that needs less number of servers.

2.2 The Virtual Machine Migration Problem

Given the minimum number *n*, the objective of VMMP is to make a placement to reduce the migration overhead as much as possible, and guarantee the feasibility of VM migration at the same time. Therefore, this problem is divided into two parts.

Which PS a single VM should migrate to is the first part of VMMP. Previous approaches [1-2] pay too much attention to whether a VM could migrate to some PS or not, ignoring the overhead that the migration itself brings. It may not be possible to migrate a VM to the destination PS immediately in the real environment, so the migration overhead plays an important role in the process of migration.

In the second part, migration constraints are taken into account. Hermenier [4] points out that the placement made by the preceding part may force a VM to migrate to some unqualified PS. In fact, the PS has the capacity to hold that VM at the end,

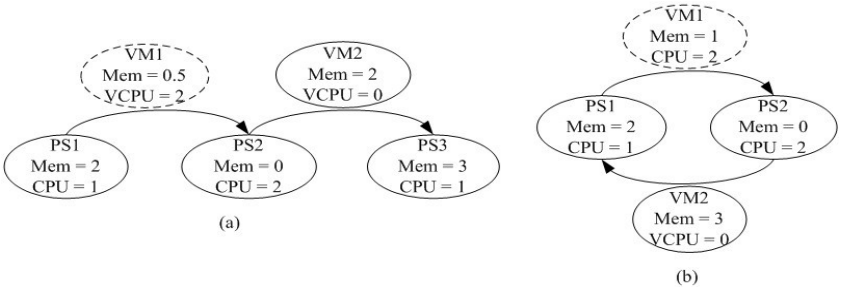

Fig. 1. Examples of migration constraints. The *Mem* and *CPU* in *PS* represent the residual resource capacity of the PS, and the *Mem* and *VCPU* in *VM* represent the resource demands of the VM. The dashed circle denotes that *VM1* is going to be activated. The *0* of *VCPU* denotes *VM2* is not active.

but it may have not enough resources in the middle of the placement. There are two types of constraints on migration: sequential constraint and cyclic constraint.

A sequential constraint takes place when one migration can not start until another migration has finished. Fig. 1(a) shows an example. Once VM1 is activated, it finds deficiency in CPU residual in PS1, so it has to migrate to PS2. However, the residual memory in PS2 can not meet the need of VM1, so a migration of VM2 from PS2 to PS3 should be executed at first. These two migrations can not occur in parallel, that is, the migration of VM1 has to wait till VM2's has completed.

A cyclic constraint occurs when multiple migrations form a cycle. Fig. 1(b) shows an example. Once VM1 is activated, it finds deficiency in CPU residual in PS1, so it has to migrate to PS2. However, the residual memory in PS2 can not meet the need of VM1, so a migration of VM2 from PS2 to PS1 is needed at first. Unfortunately, PS1 has not enough memory for VM2, which requires that the migration of VM1 should be done in the first place. These two migrations can not happen because the two VMs both lock the resources that the other need.

These two types of constraints have severe influence in the establishment of the placement. The migration can not be executed unless these constraints are removed, and the feasibility of migration is greatly threatened. Most of previous approaches [1-2, 7] may not encounter the migration constraints, because they guarantee the feasibility in each step of migration. However, as mentioned above, the locally optimal solution they adopt can not achieve a global optimization. When a migration is not feasible, Wood07 [8] identifies a set of VMs to swap to free resources on the destination server. However, this method temporarily needs some space for hosting VMs and the placement issue it is able to solve can not be complex. Girt [9] take an offline strategy in which they suspend the VMs that need migrate till the destination PS is qualified. Yet, this does not work for live migration.

3 Design and Implementation of VSCM

A typical virtual cluster consists of one management server without virtual environment and multiple computing servers (PSs as mentioned above) with Xen [10] virtual environment in which the VMs are placed. System architecture of VSCM is

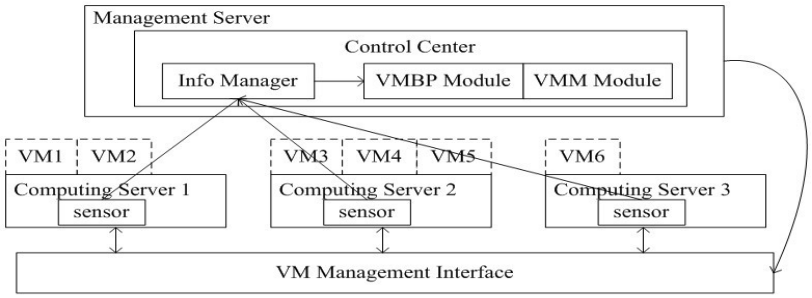

Fig. 2. VSCM system architecture. Modules of *VMBP* and *VMM* constitute the core of VSCM and the *sensor* and *Info Manager* are used to collect and arrange the info from VMs and PSs.

showed in Fig. 2. VSCM is made up of one *Control Center* in the *Management Server* and one *sensor* in each *Computing Server*. These sensors and the *Info Manager* in the *Control Center* are components of Prospector [11], a Xen virtual cluster monitor, which is used to iteratively collect the measurements of resources in PSs and VMs and detect the change of their state. After arranged by *Info Manager*, the information that is collected by sensors is sent to the core of VSCM, *VMBP* and *VMM*, two modules in the *Control Center*. *VMBP Module* uses GOVMBP algorithm to solve VMBPP and then *VMM Module* uses FVMM algorithm to solve VMMP. Once the core finds the current placement not viable, or new placement needs fewer servers, VSCM will call the *VM Management Interface* to execute the consolidation corresponding to the new plan.

3.1 The Globally Optimal Virtual Machine Bin Packing Algorithm

To dig out the possible opportunities to reduce the minimum number *n*, VMBP module uses a globally optimal algorithm, the GOVMBP, to solve the VMBPP. Making a new placement according to the current one is the limitation of FFD, so the essence of GOVMBP is to initialize the placement no matter what the current one is. That is, at the beginning of determining the placement, GOVMBP ignores the current placement, and hypothesizes that each PS is in its initial state, without any VM. Therefore, when considering the placement, GOVMBP faces a brand new cluster instead of complicated placement, which helps to completely take demands of all VMs and residuals of all PSs into account from a globally optimal view before making the decision.

GOVMBP firstly takes current placement and measurements of VMs and PSs as input, and save the current placement as *begin_cluster*. The next step, the kernel of this algorithm, is to initialize the placement to the situation in which there are no VMs on any PS. The following part is based on FFD. Firstly, on the basis of latest resource measurements of VMs and PSs, ranks the VMs in the decreasing order of resource demand; then, for each VM, if the residual of PS is enough for the VM, the VM will stay, if not, migrate this VM to the first PS which can hold it; if there's no active PS that can be migrated to, activate a new PS to hold the VM. Finally, we get the new placement *mid_cluster* as the output.

As initializing the placement, GOVMBP is no longer limited to the current placement, and finds more possibilities to reduce the number *n*, the number of active servers in new placement. However, this algorithm introduces migration constraints that are mentioned in Section 2.2 and we are going to solve them in Section 3.2.2.

There's another point to emphasize that, *mid_cluster* is just a plan which is not executed immediately, and the real placement in the cluster does not change.

3.2 Feedback based Virtual Machine Migration Algorithm

Given the minimum number *n*, VMM module adopts FVMM algorithm to solve the VMMP. Taking current placement as feedback, FVMM improves the new placement

made by GOVMBP to reduce the migration overhead, and then removes the migration constraints introduced by GOVMBP to guarantee the feasibility of the migration.

The Feedback Mechanism. Previous approaches [1-2, 7] focus on the right PS that the VM should be moved to, ignoring the migration overhead. Therefore, to reduce the migration overhead as much as possible, the feedback mechanism (FM) tries its best to avoid the migration. So, the essence of this mechanism is that, if a migration is avoidable, avoid it. After comparing the new placement with the current one, FM tries to restore the placement. That is, for each migration in the new placement, if canceling this move still guarantees that the PS has the capacity to hold the VMs and the number n is not increased, this migration will be dropped. The final objective of FM is to make every VM stay on the PS where it is placed now.

The FM works as follows. A VM that needs to move (MVM) is the one whose host server in the new placement *new_host* is not the same one in the current placement *old_host*. For each MVM v, the FM will make a judgment whether v could stay on the *old_host* or not by the analysis of the resource measurements of all the VMs and PSs in the new placement. If the following situations happen, v is allowed to stay, and the migration for v in the new placement is cancelled.

(1) The *old_host* has enough resource residual to hold v

(2) The *old_host* could move some MVMs that are hosted on it to other PSs to free the resources to hold v.

(3) Some MVM that are hosted on *old_host* could exchange the position with v, without exceeding the resources limits on both servers.

If, unfortunately, above situations do not happen, v has to move from *old_host* to *new_host*. The FM will produce a new placement *end_cluster* in which more VMs are hosted on its *old_host* than *mid_cluster*. This leads to fewer migrations and the overhead will be reduced greatly.

The *end_cluster* is the final placement which will be the real placement after this process of consolidation is done. The migrations in *end_cluster* can not be avoided and they are put on a list *move_list*. Like *mid_cluster*, *end_cluster* is not executed immediately and the real placement at this moment in the cluster does not change.

Methods to Remove Migration Constraints. As GOVMBP introduces migration constraints and the FM aggravates this situation, the feasibility of migration in the latest placement can not be guaranteed. As mentioned above in Section 2.2, there are two types of constraints on migration.

For sequential constraints, the solution lies in the description of this problem. One move M_1 can not be executed until another one M_2 has finished. As a result, finishing M_2 and M_1 in order will remove this constraint. So, for all migrations in *move_list*, search the feasible ones and remove them from the list. These migrations are executed earlier and free the resources for the ones still on the list. Repeat this process till there are no feasible migrations, and those left on the list are limited by cyclic constraints. The feasible migrations found in the same iteration can be done in parallel because there are no longer constraints between them.

For cyclic constraints, we give the following definition.

A VM vm_i needs to move from server s_i to server s_j while another VM vm_j needs to move from s_j to s_i. If the resource limits both moves, we define this situation as a (s_i, s_j, s_i) cycle. By extension, if vm_j needs to move from s_j to s_k while VM vm_k needs to move from s_k to s_i, this forms a (s_i, s_j, s_k, s_i) cycle.

According to above definition, a cyclic constraint is hard to find, and we need find a way to recognize the cycle by using Graph Theory. Given k PSs, we construct a digraph $G = (V, E)$. Vertex v_i denotes server s_i, and vertex set $V = (v_0, v_1, \ldots, v_{k-1})$ is the set of all vertexes. A directed edge $e_{i,j} = \langle v_i, v_j \rangle$ from v_i to v_j denotes that there is a VM that needs to move from v_i to v_j and edge set $E = (e_{0,1}, e_{1,2}, \ldots, e_{k-1,k-2})$ is the set of all directed edges. If there is directed edge $e_{i,j} = \langle v_i, v_j \rangle$, we define v_j as the neighbour of v_i. We also call $v_i e_{i,i+1} v_{i+1} e_{i+1,i+2} v_{i+2} \ldots e_{k-1,k} v_k$ a directed path that connect v_i to v_k. If there is a directed path $v_i e_{i,i+1} v_{i+1} e_{i+1,i+2} v_{i+2} \ldots e_{k-1,k} v_k$ from v_i to v_k, we say v_k is accessible from v_i. So, the cycle (s_i, s_j, s_i) denotes the directed path that goes from v_i to v_j by the directed edge $e_{i,j}$ and then back to v_i by $e_{j,i}$, that is, v_i is accessible from v_i itself. We define this path from v_i back to v_i as a directed cycle (DC) $v_i e_{i,j} v_j e_{j,i} v_i$. Therefore, to recognize a cycle is to recognize the corresponding DC in the graph G. For example, there are two DCs in Fig. 3 which are $v_1 e_{1,2} v_2 e_{2,5} v_5 e_{5,6} v_6 e_{6,1} v_1$ and $v_3 e_{3,4} v_4 e_{4,3} v_3$ respectively.

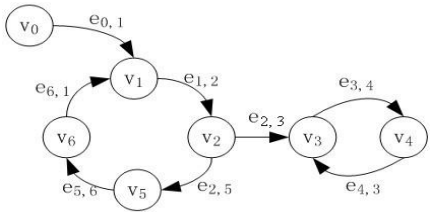

Fig. 3. An example of directed graph G in which there are two DCs, $v_1 e_{1,2} v_2 e_{2,5} v_5 e_{5,6} v_6 e_{6,1} v_1$ and $v_3 e_{3,4} v_4 e_{4,3} v_3$.

This paper introduces a method to recognize the DC in graph G on the basis of Depth First Search (DFS) algorithm. The essence of this method is that if v_i is in a DC $v_i e_{i,j} v_j e_{j,k} v_k \ldots e_{m,i} v_i$, v_i is accessible from v_i itself. So, in the graph G' which is built by cutting the edge $e_{i,j}$, there is certainly a path from v_j to v_i, that is, v_i is accessible from v_j. Furthermore, in the graph G'' built from G' by cutting the edge $e_{j,k}$, v_i is accessible from v_k. Taking advantage of DFS, we can test a vertex v whether it is accessible from itself. If accessible, v is in a DC that includes v.

Pseudo-code of the method to test whether vertex *end* is accessible from vertex *begin* on the basis of DFS.

```
1  def test_access(begin, end):
2    for v in begin.neighbours:
3      if v.des == end:
4        add begin into cycle
5        return True
6      if v.des not visited:
7        test_access(v.des, end)
```

The above pseudo-code tests whether vertex *end* is accessible from vertex *begin* on the basis of DFS. For each vertex v in the neighbors of *begin*, if its destination node

v.des happens to be *end*, *end* is accessible from *begin*. If not, test whether *end* is accessible from *v.des* on the premise that *v.des* has not been visited before.

Take Fig. 3 as an example. Starting with v_0, the search sequence is v_0-> v_1-> v_2-> v_3-> v_4, and then we find a unique path from v_4 to v_3, so we step back to v_2. However, we will soon find that, after v_2-> v_5-> v_6, v_6 can only reach v_1. That is, v_0 is not accessible from itself. If we start with v_1, we will recognize a DC $v_1e_{1,2}v_2e_{2,5}v_5e_{5,6}v_6e_{6,1}v_1$. Similarly, another DC $v_3e_{3,4}v_4e_{4,3}v_3$ is found if we take v_3 as the beginning vertex.

The next step is to untie this cycle, that is, to remove this cyclic restraint. Comparing with the method to recognize the cycle, the one to untie it is much simpler. Search all servers and find one to be a springboard. That is, for each MVM *vm* in this cycle, if some server *s* can be found enough resource residual to hold *vm*, we firstly move *vm* to *s* and then move it from *s* to its original destination. This server *s* is used as a springboard to hold *vm* temporarily.

During the process to remove the migration constraints, VSCM generates feasible migration commands for consolidation and calls the VM management interface to execute them. After the consolidation is done, the real placement in the cluster is finally changed to *end_cluster*.

4 Evaluation

This section presents results from a number of experiments we conducted to evaluate our algorithms.

4.1. Design of Migration Overhead

As one of the simplest overhead, migration step is the number of the migrations that the consolidation needs to change the current placement to the new one, directly evaluating various migration algorithms. However, resource usages on servers play a great role in moving one VM from source server to destination server. The higher the CPU or memory usage is, the longer the process of migration takes, for example. And the most important factor is the memory requirement of VM which determine the time of the process.

Therefore, we design the migration overhead as follows.

The total overhead of placement *O(p)* is the sum of each migration overhead *O(v)* made by VM *v* moving from source server s_{src} to destination server s_{des} in this migration. (Equation 1)

$$O(p)=\sum_{v\in p} O(v) \quad . \tag{1}$$

As involving the source server and destination server, *O(v)* is made up of the overhead to source $f(s_{src},v)$ and the overhead to destination $f(s_{des},v)$. (Equation 2)

$$O(v) = f(s_{src}, v) + f(s_{des}, v) \; . \tag{2}$$

In consideration of the only focus on CPU and memory in Section 2.1, the overhead *f(s, v)* made by VM *v* to server *s* is determined by following factors: CPU usage of the server *s.cpu_util*, memory usage of the server *s.mem_util* and the memory requirement of the VM *v.mem*. Since *v.mem* dominates the time of migration process, we construct Equation 3 as follows.

$$f(s, v) = (s.cpu\_util + s.mem\_util) * v.mem . \quad (3)$$

4.2. Experimental Setup

Limited by real environment, we have to evaluate our algorithms on simulation data, to illustrate the range of benefit that VSCM can provide. The virtual cluster we simulate consists of 64 PSs and 100 VMs that are generated randomly. Each PS has 4 CPU cores and 4 GB of memory. For each VM, we generate the VCPU demand randomly in the range of [0, 4] and the memory demand in the range of (0.0, 2.0] GB. For simple, the memory is going to be 0.5 or 1.0 or 1.5 or 2.0GB.

The computer configuration for this simulation is shown as follows: Intel(R) Core(TM)2 Duo CPU P8400@2.26GHz, 2x1G DDR3 memory and the operating system is CentOS 5.3. VSCM is implemented in this OS by Python 2.4.3.

4.3. Results and Analysis

GOVMBP vs FFD. To test whether GOVMBP uses fewer PSs, we compare it with FFD which is used by most of the previous approaches. The initial cluster is made up of 64 PSs without any VM and then we randomly generate 100 VMs. For the initial placement, both the algorithms adopt the same policies to allocate these VMs to the PSs, and the number of PSs in use is the same. Then we change the configuration of the 100 VMs to simulate the various situations in the real environment. Facing to the new placements in which there are already 100 VMs hosted on the PSs, GOVMBP and FFD will take different policies. As a result, the number of PSs used may not be the same, and neither do the new placements the algorithms produce.

Fig. 4 shows the comparison between GOVMBP and FFD in 20 experiments. The numbers on the right denotes the numbers of servers that GOVMBP saves. We divide them into four classes and find that, in most cases (45%), GOVMBP uses 6-10 fewer servers than FFD. Furthermore, in the vast majority of cases (95%), GOVMBP uses fewer servers than FFD. On the basis of statistics, we find that GOVMBP uses 5.9 fewer servers than FFD and saves 10.22% of servers in average, verifying the superiority of this globally optimal solution.

Effect of the Feedback Mechanism. The FM is the kernel of FVMM, so we conduct this experiment to evaluate the effect of the FM on the consolidation. In this comparison, one is the VSCM with FM and the other one is without FM. For the initial cluster where there are 64 PSs without any VM, we randomly generate 100 VMs. Without preceding placement to feed back, VSCM with FM adopt the same policy to allocate VMs to PSs as the one without FM does. Then we change the configuration of the 100 VMs to simulate the various situations in the real

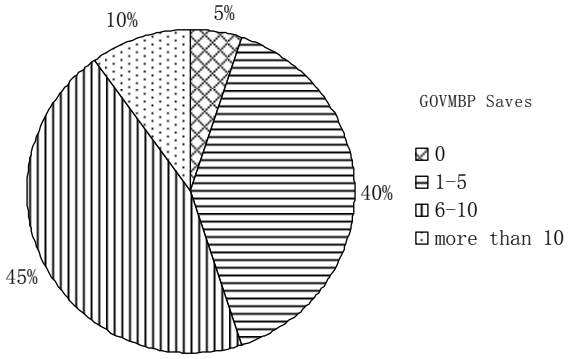

Fig. 4. Comparison between GOVMBP and FFD. The numbers on the right represents the numbers of servers that GOVMBP saves.

environment. With current placement as the feedback, the one with FM and the other one without FM will take different policies. As a result, the overhead which is produced by all the migrations will be different.

Fig. 5 shows the comparison migration overhead between VSCM with FM and the one without FM in 20 experiments. The left part compares migration step. In average, the one with FM needs 34.25 steps to finish the placement while the one without FM needs 106.25 steps which is 3.1 times the former. The right part compares the migration overhead designed by Section 4.1. The result also verifies the benefit of the FM that, the overhead produced by the one without FM is 1.5 times that produced by

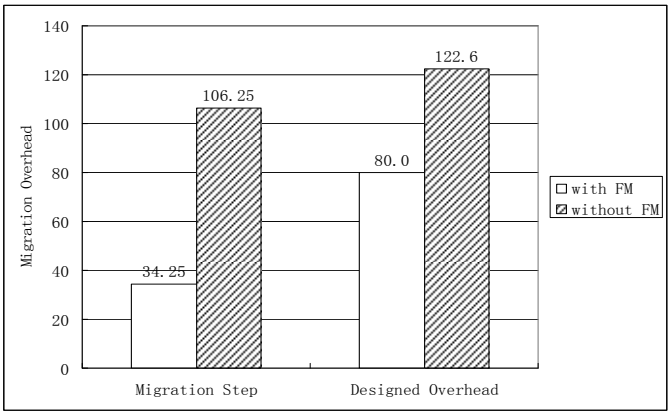

Fig. 5. Migration overhead comparison between VSCM with FM and the one without FM. The left part compares the average migration step and the right part compares the average migration overhead designed by Section 4.1.

the one with FM. According to the statistics, the FM is able to save 67.6% of migration step and 34.7% of migration overhead designed by Section 4.1.

5 Related Work

Most of previous approaches use heuristics from a locally optimal perspective to solve VMBPP, so they lose many opportunities to reduce the number of servers, and they often ignore the migration overhead which is very important for the real cluster.

Bobroff et al. [1] establishes the virtual server consolidation on a forecast mechanism. They predict the resource demands of VMs on the basis of history, rank the VMs in the decreasing order of the demands, and then use FFD to allocate VMs to PSs in arranged order. For each server, they take the sum of all the VMs' resource demands as the predicted resource overload. If this overload does not exceed the threshold of the server and the resource capacity is enough to handle the resource demand, they allocate the VM to the most suitable server. Strictly speaking, the placement algorithm they use is based on BFD.

Though Khanna et al. [12] use heuristics to solve VMBPP, it is gratifying that they pay more attention to the migration overhead while determining the placement. They rank VMs in non-decreasing order of the resource usages of VMs from which migration overhead is calculated. In other words, they rank the VMs in the order of the overhead. They choose the VM whose usage is the lowest on the unqualified PS, and move it to the qualified PS which has the least resource residuals. Wood07 et al. [8] also use a greedy algorithm to determine a sequence of moves to migrate overloaded VMs to underloaded servers, in which the migration overhead is considered in the form of defined volume. However, these approaches still base their placement algorithm on the locally optimal solution.

Some approaches mention the migration constraints in their work. Girt et al. [9] try to solve the VMMP with the help of Shirako [13], a system for on-demand leasing of shared networked resources in federated clusters. When a migration can not proceed because of the migration constraint, the VM will be suspended till the destination server is available. This simple method is not qualified in the real-time environment where needs live migration. Wood07 et al. [8] identify a set of VMs to swap to free resources on the destination server. This approach is only able to solve simple placement issue and needs some space for hosting VMs temporarily.

Unlike the above approaches, Hermenier et al. [4] develop the Entropy by Choco [14] to perform consolidation based on constraint programming. From the globally optimal point of view, Entropy solves VMBPP and VMMP pretty well. Comparing to FFD, Entropy greatly reduces the number of servers and the migration overhead. However, Entropy aims at homogenous cluster and one PS is allowed to run only one VM at a time. This is much different from the real cluster of data center where there are usually many VMs placed on one PS. As a result, using CP in a real cluster environment will produce significant time overhead that data center can not afford.

6 Conclusions

Virtual server consolidation which can reduce large number of servers helps the enterprises reduce hardware and operating costs significantly and improve server utilization greatly in the enterprise-class data center.

In this paper, we propose a new approach to virtual server consolidation that considers both the problem of allocating the VMs to available PSs and the problem of how to migrate the VMs to these servers. Experiment results demonstrate that, our consolidation manager VSCM can indeed reduce the number of servers greatly, as compared to previously used FFD. While reducing the migration overhead significantly, VSCM also guarantees the feasibility of each migration operation.

In future work, we plan to improve the Globally Optimal Virtual Migration Bin Packing algorithm on the basis of more complicated bin-packing algorithm. We also expect to apply VSCM to the real cluster to get convincing experiment results.

References

1. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In IEEE Conf. Integrated Network Management, (2007)119-128
2. Hyser, C., Mckee, B., Gardner, R., Watson, B.J.: Autonomic Virtual Machine Placement in the Data Center. Technical Report HPL-2007-189, HP Labs, (2007)
3. Benhamou, F., Jussien, N., O'Sullivan, B.: Trends in Constraint Programming. ISTE, London (2007)
4. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a Consolidation Manager for Clusters. In Proceedings of the ACM/Usenix International Conference On Virtual Execution Environments (VEE'09), (2009)41-50
5. Coffman, Jr.E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin-packing - An updated survey. Approximation Algorithms for Computer System Design, (1984)49-106
6. Lodi, A., Martello, S., Vigo, D.: Recent advances on two-dimensional bin packing problems. Discrete Appl. Math., Vol. 123(1-3). (2002)379-396
7. Verma, A., Ahuja, P., Neogi, A.: Power-aware dynamic placement of HPC applications. Proceedings of the 22nd annual international conference on Supercomputing (ICS'08), (2008)175-184
8. Wood, T., Prashant, S., Arun, V., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In Proceedings of the Fourth Symposium on Networked Systems Design and Implementation (NSDI), (2007)229-242
9. Grit, L., Irwin, D., Yumerefendi, A., Chase, J.: Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration. In Proc. VTDC'06, (2006)1-8
10. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauery, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. Proc. of ACM SOSP, Vol. 37(5). (2003)164-177
11. Liu, Y., Xiao, N., Shen, Li.: Design and Implementation of Xen Virtual Cluster Monitor. Jounal of Wuhan University of Technology, Vol. 32(20). (2010)184-188
12. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application Performance Management in Virtualized Server Environments. Network Operations and Management Symposium, (2006)373-381
13. Irwin, D., Chase, J., Grit, L., Yumerefendi, A., Becker, D.: Sharing networked resources with brokered leases. ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference, USENIX Association (2006)18-18
14. Jussien, N., Rochart, G., Lorca, X.: The CHOCO constraint programming solver. CPAIOR'08 Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP), Paris (2008)