Adaptive Resource Provisioning for the Cloud Using Online Bin Packing

Weijia Song, Zhen Xiao, *Senior Member, IEEE,* Qi Chen, and Haipeng Luo

Abstract—Data center applications present significant opportunities for multiplexing server resources. Virtualization technology makes it easy to move running application across physical machines. In this paper, we present an approach that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers actively used. We abstract this as a variant of the relaxed on-line bin packing problem and develop a practical, efficient algorithm that works well in a real system. We adjust the resources available to each VM both within and across physical servers. Extensive simulation and experiment results demonstrate that our system achieves good performance compared to the existing work.

Index Terms—Cloud Computing, virtualization, green computing.

1 Introduction

Amazon EC2 provides networked virtual machines (VM) for lease. This model, named as Infrastructure as a Service (IaaS), is broadly accepted in the industry. It is estimated that the number of daily VM launches in the "us-east-1" region has grown from 6,738 in year 2008 to 70,528 in year 2010 [1]. The total number of physical servers is close to half a million in 2012 [2].

The load of applications encapsulated in the VMs may change dynamically. The resource demands of applications can be highly variable due to the time of the day effect, sudden surge in user requests, and other reasons. Moreover, some applications such as on-demand data processing and virtual desktop have random load. It is hard to forecast their resource demands. Although server consolidation has the effect of absorbing load fluctuation, the sum of the peak resource demand of VMs sharing a physical machine (PM) can be much higher than the mean. Thus, it is often inefficient to over-provision resources based on the peak demand because doing so will leave the resources under-utilized most of the time.

Live migration [3] allows a VM to be moved from one server to another without interrupting the application running inside. This can be used to adjust the VM layout for load balancing or energy saving purpose. For example, when the resource utilization of a server becomes too high, some VMs running on it can be migrated away to reduce its load. On the contrary, when the average server utilization in the system becomes too low, the VMs can be aggregated to a fraction of the servers so that idle servers can be put to sleep to save power. Previous work has shown that energy is a significant part of the operational cost in data centers [4] and that it takes a server only several seconds to sleep or to wake up [5] [6]. Novel hardware assisted servers can accomplish the transition between sleep and active states in negligible milliseconds [7]. Compared with application layer solutions [4] [8], this approach enjoys the simplicity of being application neutral. However, live migration incurs overhead and may last some period of time depending on the workload inside the VM and the concurrent network traffic.

Various migration policies have been designed in the literature [9] [10] [11] [12] [13] [14] [15]. In Sandpiper [9], a server whose resource utilization is above some predefined thresholds is called a "hot spot". Its black box/gray box (BG) scheduling algorithm periodically detects hot spots in the system and decides which VMs should be migrated away to relieve the hot spots. It does not consolidate VMs at valley load for green computing purpose, since its main focus is overload avoidance. Nor does VectorDot [11] perform green computing.

The resource allocation problem can be modeled as the bin packing problem where each server is a bin and each VM is an item to be packed. The approach presented in [10] periodically performs an offline bin packing algorithm to calculate a new VM layout, where hot spots are eliminated and the number of servers in use is minimized. VMs are then migrated accordingly. The new layout, however, may be quite different from the existing one, which may incur a large number of VM migrations. Wang et al. adopts an online bin packing algorithm which incurs fewer migrations [12]. The focus of that work is on the network bandwidth only and does not consider multi-dimensional resource constraints.

In this paper, we present an approach that uses live migration to allocate data center resources dynamically based on application demands and supports green

- *W. Song, Z. Xiao, and Q. Chen are with the Department of Computer Science at Peking University. Z. Xiao is the contact author. E-mail: {songweijia,xiaozhen}@pku.edu.cn, chenqi@net.pku.edu.cn*
- *H. Luo is with the Department of Computer Science at Princeton University, Princeton, New Jersey. Email: haipengl@cs.princeton.edu*

Digital Object Indentifier 10.1109/TC.2013.148

computing by optimizing the number of servers used. We make the following contributions:

- We develop a practical bin packing resource allocation algorithm. It is capable of both overload avoidance and green computing. It is applicable to scenarios with multi-dimensional resource constraints.
- We give a theoretical proof that the number of servers in use is bounded by $3/2 * d$, where d is the number of bottleneck resources.[1] Especially, with only one bottleneck resource, the number of servers in use is bounded by 1.5 times of the optimal value. We also prove that the number of VM migrations is bounded as well.
- We conduct extensive simulation and experiments. The results demonstrate good performance compared with the existing work.

The rest of the paper is organized as follows. Section 2 gives an overview of our approach. Section 3 describes our design, bin packing with variable item size (VISBP) algorithm. Section 4 and 5 present simulation and experiment results, respectively. Related work is described in Section 6. Section 7 concludes.

2 Overview

Figure 1 gives an overview of a typical virtualized data center where our approach applies. The components of our approach are marked with gray color. Each physical machine (PM) runs a virtual machine monitor (VMM) like Xen [17], KVM [18] or VMware [19]. Multiple virtual machines (VM) share a PM. Applications such as Web server, remote desktop, DNS, Mail server, Map/Reduce, etc., run inside the VMs. The VMM collects the usage statistics of resources on corresponding PM such as CPU utilization, memory consumption, network sends and receives, etc.. These statistics are forwarded to a centralized VM scheduler, the key component that is responsible for load balance and green computing by adjusting the mapping of VMs to PMs.

The VM Scheduler is invoked periodically and receives the following inputs:

- the resource demand history of VMs
- the capacity and the load history of PMs
- the current assignment of VMs to PMs

The VM Scheduler has two modules. The load predictor estimates the resource demands in the near future. The scheduling algorithm optimizes VM layout so that the resource demands are satisfied and resource waste is minimized. Although the VM scheduler may seem to be a single point of failure, it is not involved in the data path during normal processing. In other words, its failure will cause the system to stop adjusting the VM to PM layout in response to load changes in the system, but the existing layout can continue functioning normally. Should a higher degree of reliability be needed, we can use the hot mirroring technology [16] or the fast restart tehcnology [17].

1. We consider the resources whose utilization are significantly higher than others as the bottleneck resources. For example, if most applications in a system are CPU-intensive, then the CPU resource is one of the bottleneck resources.

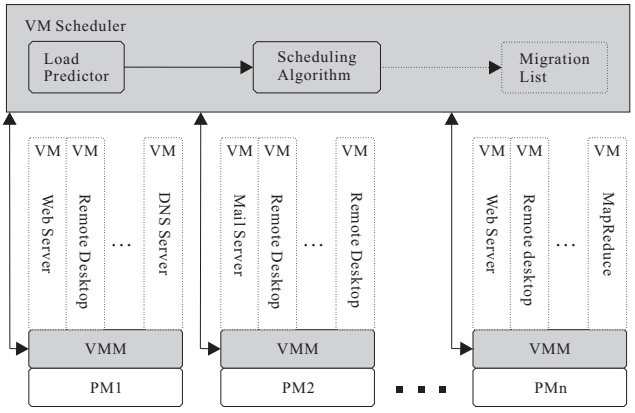


Fig. 1. Overview of the System Architecture

The VM Scheduler is invoked periodically. In each round, the load predictor estimates the resource demands of all applications (VMs) and the load of all PMs based on past statistics. The estimates are represented by the number of CPU ticks, the amount of memory, and the bandwidth of network I/O. Based on the estimates, the scheduling algorithm checks whether every PM has enough resources for the appointed demands. If so, the VMM is able to adjust resource allocation locally. Otherwise, the algorithm performs overload avoidance by migrating away some of the VMs on the overloaded PMs. The scheduling algorithm also consolidates VMs on the under-utilized PMs so that some of the PMs become idle and can be put into the standby mode to save energy. The output of the scheduling algorithm is a VM migration list which is then dispatched to the VMM on the involved PMs for execution. We assume that all PMs share a back-end storage.

Using live migration as a tool for resource allocation seems challenging at first glance. But this classical architecture is broadly adopted in the literature [9] [11] [18] [10]. Clark et al. also demonstrated that live migration incurs only negligible overhead on applications [3]. We will show that it is practical to use live migration in our system.

We use a variant of the exponentially weighted moving average (EWMA) load predictor. The ESX Server uses traditional EWMA formula like the following:

$$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t), 0 \leq \alpha \leq 1$$

where E(t) and O(t) are the estimated and the observed load at time t, respectively. α reflects a tradeoff between stability and responsiveness. ESX computes three EWMA estimates with different parameters and takes the maximum of them as a conservative estimate on the VM's memory needs. Although seemingly satisfactory, it does not capture the rising trends of resource usage. For example, when we see a sequence of $O(t) = 10, 20, 30,$ and 40, it is reasonable to predict the next value to be 50. Unfortunately, when α is between 0 and 1, the predicted value is always between the historic value and the observed one. To reflect the

"acceleration", we take an innovative approach by setting α to a negative value. When $-1 \leq \alpha < 0$, the above formula can be transformed into the following:

$$\begin{aligned} E(t) &= -|\alpha| * E(t-1) + (1+|\alpha|) * O(t) \\ &= O(t) + |\alpha| * (O(t) - E(t-1)) \end{aligned}$$

On the other hand, when the observed resource usage is going down, we want to be conservative in reducing our estimate. Hence, we use two parameters, $\uparrow \alpha$ and $\downarrow \alpha$, to control how quickly E(t) adapts to changes when O(t) is increasing and decreasing, respectively. We call this the FUSD (Fast Up and Slow Down) algorithm.

So far we take O(t) as the last observed value. Most applications have their SLAs specified in terms of a certain percentiles of requests meeting a specific performance level. More generally, we keep a window of W recently observed values and take O(t) as a high percentile of them. As we will see later in the paper, the prediction algorithm plays an important role in improving the stability and performance of our resource allocation decisions.

3 Resource allocation as Bin Packing

The classical bin packing problem consists of packing a series of items with sizes in the interval (0, 1] into a minimum number of bins with capacity one. We can model resource allocation as the bin packing problem where each PM is a bin and each VM is an item to be packed. We assume that all PMs are homogeneous with unit capacity. We normalize the resource demands of VMs to be a fraction of that capacity. For example, if a VM requires 20% of the physical memory of the underlying PM, then it corresponds to an item with size 0.2.

It is well-known that the problem is NP-hard. The quality of a polynomial time approximation algorithm A is measured by its approximation ratio $R(A)$ to the optimal algorithm OPT:

$$R(A) = \lim_{n\to\infty} \sup_{OPT(L)=n} \frac{A(L)}{OPT(L)} \qquad (1)$$

where L is the list of the input sequence and $A(L)$ and $OPT(L)$ are the number of bins used under the A algorithm and the optimal algorithm, respectively [19].

Although the bin packing problem has been studied extensively in the literature, the existing solutions do not work well in data center environments. Offline algorithms can achieve an approximation ratio very close to one [19]. This naturally leads to an approach that periodically invokes an offline algorithm to adjust the VM layout. The drawback of this approach is that it may incur a large number of VM movements when the load of VMs changes dynamically because an offline algorithm by its nature does not consider the existing VM layout when packing the VMs into the set of PMs.

There are also online bin packing algorithms which pack the current item without knowledge of subsequent items. Strict online algorithms do not permit moving any previously packed item and have a theoretical lower bound of 1.536 for the approximation ratio [20]. This is overly restrictive in our context since virtualization technology enables VM migrations in real time. As we will see later, doing so allows us to achieve a better approximation ratio even though we do not attempt to pack the bins nearly as full.

Close to our work are the existing relaxed online algorithms [21] which allow a constant number of movements of already packed items in response to the arrival of a new item. Unfortunately, those algorithms are not applicable to our settings either, because they do not support the size changes of already packed items. Note that the resource demands of VMs can change over time (which motivated us to multiplex data center resources in the first place), the sizes of items in our bin packing problem are not fixed. One might think that we can handle the size change of a previously packed item by removing it from the bin and repack it, since item removal can be supported by the delete operation in dynamic bin packing algorithms [22] [23] [24]. Unfortunately, it is easy to construct counterexamples where the correct strategy is to repack some other items in that bin instead of the changed item. In other words, when the resource demand of a VM changes, we may decide to migrate some other VMs sharing the same PM instead of migrating the changed VM. One solution is to repack all items in the changed bin, but doing so causes too many movements and defeats the purpose of an online algorithm. Moreover, many existing algorithms work by maintaining certain properties of the used bins (to keep them sufficiently full in order to prove the approximation bound). Removing an item (or reducing its size) can break those properties, leading to the reshuffle of many other items (including those from the unchanged bins). In short, the size change of an item cannot be accommodated in the existing algorithms easily.

In order to handle the changing resource demand of a VM, we design a relaxed online bin packing algorithm called Variable Item Size Bin Packing, or VISBP. It features carefully constructed categories of items and bins. Moderate size change can be absorbed as long as the category rules are kept. It is important to realize that our design principle is different in the face of the highly variable resource demands of data center applications. While the classical bin packing algorithms (online or not) consider packing a bin completely full a success, in data center environments keeping servers running at 100% utilization is detrimental to the stability of the system. In the following, we first describe our algorithm in the one dimensional case and then extend it to multi-dimensional.

3.1 One dimensional VISBP

The key idea of VISBP is to trade the approximation ratio for system stability. We divide items into four types according to their sizes. We restrict the combinations of item types in a bin to minimize the amount of space wasted.

- **insert(item)**: putting *item* into some bin
- **change(olditem, item)**: an already packed *olditem* has become a new *item* (differ only in size)

The VISBP is relaxed online in that (1) no knowledge of the future is assumed when handling the current event; (2) a small number of movements of already packed items are allowed. It differs from the existing (relaxed) online algorithms in the support of the **change** operation. The **insert** operation is similar to [21]. In the following, we present an algorithm with an approximation ratio of 1.5 which limits the number of movements for **insert** to at most 3 and for **change** to at most 7. We start with some definitions.

3.2 Definitions

Definition 1:

- Let b be a bin. gap(b) denotes the space available in bin b.
- Let i be an item. size(i) denotes the size of the item (always $\in (0, 1]$).
- Let g be a group (see Definition 4 below). size(g) denotes the size of the group, which is the total size of all items in that group.
- Let x be an item or a group. bin(x) denotes the bin it is in.

Definition 2: we divide items into several types based on their sizes.

- T-item (tiny): size $\in (0, 1/3]$
- S-item (small): size $\in (1/3, 1/2]$
- L-item (large): size $\in (1/2, 2/3]$
- B-item (big): size $\in (2/3, 1]$

The above classification of items facilitates the construction of a bin whose gap is less than 1/3 such as a bin with two T items or one B-item. The VISBP algorithm tries to maintain this property for all bins so that its approximation ratio is bounded.

Definition 3: we divide bins into several types based on their content.

- B-bin: has only one B-item.
- L-bin: has only one L-item.
- LT-bin: has only one L-item and a certain number of T-items.
- S-bin: has only one S-item.
- SS-bin: has only two S-items.
- LS-bin: has only one S-item and one L-item.
- T-bin: has only a certain number of T-items. It is called *unfilled* if the available space is no less than 1/3. Otherwise, it is called *filled*.

Definition 4: We group a certain number of T-items in the same bin into a group. All T-items in a bin form several non-overlapping groups such that: 1) the size of any group is no more than 1/3, and 2) the size of any two groups is larger than 1/3.

This grouping is convenient in that those tiny items in a group can be considered as a whole and moved in a single step. It is also reasonable since the overhead of VM migration depends mostly on the size of its memory footprint (assume shared network storage). Hence, migrating a large VM can have a comparable overhead to the migration of several small ones.

Some notations: the item and bin types in the following are examples only. Other types are interpreted in a similar way.

- $x \in$ T-item: x is a T-item.
- $x \in b$: x is an item in bin b.
- $b \in$ S-bin: b is an S-bin.
- $g \in b$: g is a group in bin b.
- $x \in$ (S-item, b): x is an S-item in bin b.
- UT-bin: unfilled T-bin.
- ULLT-bin: L-bin or LT-bin and the available space is no less than 1/3 (i.e., gap $\geq 1/3$).

3.3 Primitive operations

Now we define the following primitive operations.

new(x): create a new bin, insert item or group x in it, and return that bin.

move(x, b): move item or group x into bin b.

hot(b): check if the total size of all items in bin b is greater than 1.

fillwith(x) where x is a T-item or group:
if $\exists$ b $\in$ ULLT-bin
then move(x, b)
else if $\exists$ b $\in$ UT-bin
    then move(x, b)
    else new(x)

fill(b) where b is a L-bin or LT-bin:
while gap(b) $\geq 1/3$ and $\exists$ t $\in$ T-bin do
  if $\exists$ ut $\in$ UT-bin
  then move(g, b), g $\in$ ut
  else move(g, b), g $\in$ t

insert_S-item(x), x $\in$ S-item:
if $\exists$ b $\in$ S-bin
then move(item, b)
else new(item)

release(b) where b is a bin:
while $\exists$ g $\in$ b do fillwith(g)

adjust(b), b $\in$ LT-bin or L-bin:
while hot(b) do fillwith(g), g $\in$ b
if gap(b) $\geq 1/3$ then fill(b);
The two statements above are mutually exclusive.

3.4 Algorithm

We are now ready to construct the **insert** and **change** operations. The two operations must keep the gap of most bins within 1/3.

insert(item)

if item $\in$ B-item then new(item);
if item $\in$ L-item then b = new(item); fill(b);
if item $\in$ S-item then insert_S-item(item);
if item $\in$ T-item then fillwith(item);

Now we describe the **change** operation which is the core of our algorithm. We use $X \rightarrow Y$ to denote the types of the item before and after the change as X and Y, respectively. For example, $B \rightarrow S$ means the item changes from a B-item to an S-item. Not all type changes requires adjustment. On the other hand, adjustment may be needed even if the type does not change. In the following, any condition "$\exists X$-bin" does not consider the bin where the change is happening.

change(olditem, item):

ob = bin(item);
$B \rightarrow L$: fill(ob)
$B \rightarrow S$: if $\exists$ b $\in$ S-bin then move(item, b);
$B \rightarrow T$: if $\exists$ b $\in$ ULLT-bin then move(item, b)
else if $\exists$ b $\in$ UT-bin then move(item, b)

$L \rightarrow B$: release(ob);
$L \rightarrow L$: adjust(ob);
$L \rightarrow S$: release(ob);
if $\exists$ b $\in$ S-bin then move(item, b);
$L \rightarrow T$: if $\exists$ T-bin
then while $\exists$ b $\in$ UT-bin do
move(g, b), g $\in$ ob
else while $\exists$ b $\in$ ULLT-bin do
move(g, b), g $\in$ ob

$S \rightarrow B$: if $\exists$ x $\in$ (S-item, ob) then insert_S-item(x);
$S \rightarrow L$: if $\exists$ x $\in$ (S-item, ob) then insert_S-item(x);
fill(ob);
$S \rightarrow T$: if $\exists$ x $\in$ (S-item, ob) and $\exists$ b $\in$ S-bin then
move(x, b);
if $\exists$ b $\in$ ULLT-bin then move(item, b)
else if $\exists$ b $\in$ UT-bin then move(item, b)
else if $\exists$ x $\in$ (S-item, ob) then new(item)

$T \rightarrow B$: if $\exists$ x $\in$ (L-item, ob) then fill(new(x));
release(ob);
$T \rightarrow L$: if $\exists$ x $\in$ (L-item, ob) and x $\neq$ item then
fill(new(x));
adjust(ob);
$T \rightarrow S$: if $\exists$ x $\in$ (L-item, ob)
then insert_S-item(item); fill(bin(x));
else if $\exists$ b $\in$ S-bin
then while $\exists$ b' $\in$ UT-bin and $\exists$ g $\in$ ob do
move(g, b');
move(item, b);
else release(ob);
$T \rightarrow T$: if $\exists$ L-item in ob
then adjust(ob);
else if hot(ob) then fillwith(item);
else while gap(ob) $\geq 1/3$ and $\exists$ b $\in$ UT-bin do
move(g, b), g $\in$ b

3.5 Analysis of the approximation ratio

Recall the definition of the approximation ratio at the beginning of the section. It is easy to prove the following Lemmas.

Lemma 1: At any time, the algorithm can generate only six types of bins: B-bin, L-bin, LT-bin, S-bin, SS-bin, and T-bin.

The LS-bin may appear under the optimal packing, but not under our algorithm because we never put an L-item and an S-item together. Note that the UT-bin and ULLT-bin mentioned in the algorithm are only syntactic sugar.

Lemma 2: At any time, if there exists a T-bin, then there is no L-bin, and the available space in any LT-bin is less than 1/3.

Lemma 3: At any time, there are at most one S-bin and at most one unfilled T-bin.

Theorem 1: The approximation ratio R(A) = 3/2.

We first prove that $R(A) \leq 3/2$. We divide the proof into two cases depending on whether there exists any T-bin.

Case 1: there exists a T-bin. From Lemma 2, we know there are only B-bin, LT-bin, S-bin, SS-bin, and T-bin in the system, and the available space in any LT-bin is less than 1/3. The available space in B-bin, SS-bin, and all filled T-bin is less than 1/3 as well. From Lemma 3, there are at most one S-bin and at most one unfilled T-bin. We can disregard these two types of bins in our analysis when OPT(L) approaches infinity.[2] Hence, $OPT(L) \geq A(L) * 2/3$, which means $R(A) \leq 3/2$.

Case 2: there is no T-bin. In this case, if we discard all T-items, A(L) does not change, and OPT(L) may become smaller or remain the same. Hence, in the following we assume there is no T-item. Again we disregard S-bin because there is at most one. Under the optimal packing, there can be at most one new type of bins – LS-bin. We define

- $N_0, N_0(A)$: the number of B-bins under the optimal and our algorithms, respectively.
- $N_1, N_1(A)$: the number of L-bins under the optimal and our algorithms, respectively.
- $N_2, N_2(A)$: the number of SS-bins under the optimal and our algorithms, respectively.
- N_3: the number of LS-bins under the optimal algorithm.

It is easy to see[3]

$$\begin{aligned} N_0 &= N_0(A) \\ N_1 + N_3 &= N_1(A) \\ 2N_2 + N_3 &= 2N_2(A) \end{aligned}$$

2. In our algorithm, there will be at most one S-bin and one unfilled T-bin. Denote the S-bin as *s* and the unfilled T-bin as *t*. We can get $5/6 < gap(t) + gap(s) < 5/3$. If $gap(t) + gap(s) < 1$, we still need two bins for items in *t* and *s*. Then we get $OPT(L) >= (A(L) - 2) * 2/3 + 2 = A(L) * 2/3 + 2/3 >= A(L) * 2/3$. Otherwise, when $gap(t) + gap(s) >= 1$, we have $OPT(L) >= (A(L) - 2) * 2/3 + 1 = A(L) * 2/3 - 1/3$. This can be written as $A(L) <= OPT(L) * 3/2 + 1/2$. Simply put, in the worst case, our algorithm may use one more bin than $OPT(L) * 3/2$.

3. A B-item can not coexists with an L-item and a S-item. The number of B-bins is equal to the number of B-items, whichever algorithm is adopted. That's how we made $N0 = N0(A)$. The other two equations can be proved likewise.

So we have

$$\begin{aligned} A(L) &= N_0(A) + N_1(A) + N_2(A) \\ &= N_0 + N_1 + N_3 + (N_2 + N_3/2) \\ &\leq 3/2 * (N_0 + N_1 + N_2 + N_3) \\ &= 3/2 * OPT(L) \end{aligned}$$

Hence, $R(A) \leq 3/2$.

Now we use a specific example to show that the bound is tight. Suppose we have $2n$ **insert(item)** requests where half of the items to be inserted have the size $0.5 + \epsilon$ and the other half have the size $0.5 - \epsilon$. Then $A(L) = 3n/2$ and $OPT(L) = n$. Hence, $R(A) = 3/2$.

Our approximation ratio is about 20% worse than the best known relaxed online algorithm (which does not support the **change** operation and incurs a very large number of movements) because we do not pack the bins nearly as full [21]. It is better than the approximation ratio of strict online algorithms [20].

One may wonder if a better approximation ratio can be achieved with a even finer division of the item types. In theory, this is possible as shown in the previous work [21]. In practice, however, doing so can incur a large number of **change** operations since a small fluctuation of VM load may change the type of the corresponding item. In a cloud computing environment where the load of the VMs may change dynamically, such practice can be detrimental to the performance and the stability of the system.

3.6 Analysis on the number of movements

Recall that each movement can move either one item or one group.

Lemma 4: From any given bin, we can move at most two groups out of it into the same L-bin.

Proof: for any three groups g_1, g_2, and g_3 in the same bin, we have

$$\begin{aligned} size(g_1) + size(g_2) &> 1/3 \\ size(g_2) + size(g_3) &> 1/3 \\ size(g_3) + size(g_1) &> 1/3 \end{aligned}$$

adding them together, we have

$$size(g_1) + size(g_2) + size(g_3) > 1/2$$

Since the available space in L-bin is less than 1/2, we cannot move all three groups into it.

Lemma 5: In any filled T-bin, there are at least three groups.

Proof: the size of a group is at most 1/3, and the available space in a filled T-bin is less than 1/3. The conclusion follows.

Lemma 6: Let g_1, g_2, g_3, and g_4 be four groups. g_1 and g_2 belong to the same bin, and g_3 and g_4 belong to the same bin. Then it is not possible to move all four of them into the same L-bin.

Proof: $size(g_1) + size(g_2) > 1/3, size(g_3) + size(g_4) > 1/3$. The available space in a L-bin is less than 2/3. The conclusion follows.

Lemma 7: There are at most five groups in a bin.

Proof: Assume for the sake of contradiction there are six groups in a bin, $g_i (1 \leq i \leq 6)$. Then we have

$$\begin{aligned} size(g_1) + size(g_2) &> 1/3 \\ size(g_3) + size(g_4) &> 1/3 \\ size(g_5) + size(g_6) &> 1/3 \end{aligned}$$

adding them together, we have $\sum_{i=1}^{6} g_i > 1$, which exceeds the capacity of the bin.

Theorem 2: the number of movements in the **insert** operation is at most 3.

Proof: Only when the newly arrived item is an L-item do we need to move already packed groups. From Lemma 4, 5, and 6, we know the worst case happens when we (1) move two groups into an unfilled T-bin, and move a group into a filled T-bin, or (2) move a group into an unfilled T-bin, and move two groups into a filled T-bin. In either case, the number of movements is at most 3.

Theorem 3: the number of movements in the **change** operation is at most 7.

Proof: based on the types of the item before and after the size change, we list the number of movements in the worst case in Table 1. The analysis is similar to the previous theorem.

TABLE 1
Number of Movements in Worst Case

Old \ New	B-item	L-item	S-item	T-item
B-item	0	3	1	1
L-item	2	3	3	3
S-item	1	4	0	2
T-item	5	7	6	5

3.7 Time complexity of VISBP

It is easy to see that each operation of VISBP can be completed in $O(\log n)$ time. Note that the best offline algorithm has a time complexity of $O(n \log n)$. Hence, our $O(\log n)$ time per operation is asymptotically the best.

3.8 Multi-dimensional VISBP

We extend VISBP to multi-dimensional by breaking down items according to their largest dimensions. More formally, let A_d be a d-dimensional bin packing algorithm with the following input and output.

Input: $L_d = \{v_1, v_2, \cdots, v_n\}$, where $v_i = (v_i^1, v_i^2, \cdots, v_i^d)$, and $0 \leq v_i^j \leq 1$, $(i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, d)$.

Output:

- Positive integer $A_d(L_d)$: total number of bins used. The bins are numbered from 1 to $A_d(L_d)$.
- $O_d = \{(1, b_1), (2, b_2), \cdots, (n, b_n)\}$, where b_i is the index of the bin for item i $(1 \leq b_i \leq A_d(L_d))$.

Let A_1 be a one-dimensional bin packing algorithm. Now we extend it to a d-dimensional algorithm A_d as follows:

For any input $L_d = \{v_1, v_2, \cdots, v_n\}$ for A_d, construct an input $L_1 = \{a_1, a_2, \cdots, a_n\}$ for A_1, where $a_i =$

$max\{v_i^1, v_i^2, \cdots, v_i^d\}$. Let the output of A_1 be $A_1(L_1)$ and O_1. Then the output of A_d is: $A_d(L_d) = A_1(L_1), O_d = O_1$. It is easy to see that this output is legitimate.

Theorem: $R_d = d * R_1$, where R_1 and R_d are the approximation ratios of A_1 and A_d, respectively.

We omit the proof here due to lack of space. Our method is very general in that it can transform any one dimensional algorithm into the corresponding multi-dimensional version. The worse case is when each item has a different bottleneck dimension. For example, when d = 3, the items (1, 0, 0), (0, 1, 0), and (0, 0, 1) can be packed into a single bin under the optimal algorithm, but need three bins under VISBP.

This ratio is rather unimpressive at a first glance. Some algorithms in the literature can achieve a much better ratio when d is genuinely large. However, those algorithms cannot be applied to our problem because they are either offline [25] or do not support the **change** operation [26]. More importantly, there are not that many bottleneck dimensions in practice. Most practical systems consider only two or three types of resources in their allocation decision. A leading commercial middleware product considers only CPU and memory when allocating resources for a group of Web applications [8]. Other types of resources are not critical to their performance (i.e., they are not the bottleneck dimensions). Sandpiper considers CPU, memory, and network resources for hot spot mitigation but then combines them into a single "Volume" metric in their VM migration decision [9]. Our own trace collection at our university has also verified that the number of bottleneck dimensions is small (see Section 4 for details). When $d = 2$ or 3, VISBP is within 11% or 22% of the best known (polynomial time) online algorithm, respectively [26]. It is possible to design a more complicated algorithm with a better approximation ratio when d is really large, but we have not found a compelling reason to do so.

4 SIMULATION

4.1 Existing algorithms for comparison

In this section, we compare VISBP with three algorithms using trace driven simulation: the Black-box and Gray-box algorithm, the VectorDot algorithm, and the offline bin packing algorithm.

The Black-box and Gray-box algorithm [9] (denoted as BG) combines utilization of CPU, memory, and network I/O bandwidth into a single "volume" metric, which quantifies the resource demands of a VM or the load of a PM. When solving a hot spot, the BG algorithm tries to migrate away a VM with the highest volume to memory size ratio to a PM with the lowest volume. This maximizes the volume brought away by each migrated byte.

In HARMONY [11], PMs are connected by data center network to a central storage management device. When a PM or a switch in the data center network becomes a hot spot, the VM that contributes the most to the hot spot is moved elsewhere. VectorDot introduces an "EVP" metric, which is calculated by the dot product of the utilization vector of a VM and that of a PM hot spot, to quantify the contribution of the VM to the hot spot. It also helps decide the migration destination. This simulation does not involve data scheduling.

For the offline bin packing algorithm [10] (denoted as offline-BP), during each scheduling interval, all VMs are sorted by the resource demand in descending order. The First-Fit heuristic is then used to calculate an approximate solution.

4.2 Design of the simulator

We evaluate the effectiveness of VISBP using trace driven simulation. We use the framework from an open source cloud management software called Usher [27]. Our simulation uses the same code base for the algorithm as the real implementation in Usher. This ensures the fidelity of our simulation results. The simulator is implemented in fewer than 4000 lines of Python code. It runs in a Linux box over a rack-mount server with two Intel E5520 CPUs and 32 Gigabytes of Memory.

We have collected traces from a variety of servers in our university including our faculty mail server, the central DNS server, the syslog server of our IT department (which processes logs collected from all major gateways on campus), the index server of our P2P storage system, and many others. We have also collected desktop traces in our department using tools like "perfmon" (Windows), the "/proc" file system (Linux), "pmstat/vmstat/netstat" commands (Solaris), etc.. We post-processe the traces based on days collected into more than 500 datasets. We use random sampling and linear combination of the datasets to generate the VM traces.

Before the VM traces are fed to the scheduling algorithms, they are all filtered by our FUSD load prediction algorithm with $\uparrow \alpha = -0.2$, $\downarrow \alpha = 0.7$, and $W = 8$. We later discuss the choice of the α parameters in Section 4.6. The ratio of the number of VMs to PMs are kept at 20:1 thoughout the simulation.

4.3 Green Computing

We simulate a system with 60 PMs and 1200 VMs. The bottom part of Figure 2 shows the daily load variation in the system. The x-axis is the time of the day starting at 8am. We call a PM *active* if it has at least one VM running. Otherwise, it can potentially be put to sleep. The two y-axes are the percentage of the load (right) or the number of Active PMs in the system (left). As can be seen from the figure, the CPU load demonstrates a diurnal pattern which decreases substantially after midnight. The memory consumption is fairly stable over the time. The network utilization stays very low and has no impact on our scheduling decision. This confirms our early observation that most practical systems have only a small number of bottleneck resources. We consider only the CPU and the memory in our simulation.

The top part of figure 2 compares how the percentage of APMs (Active PMs) changes along with the workload. It

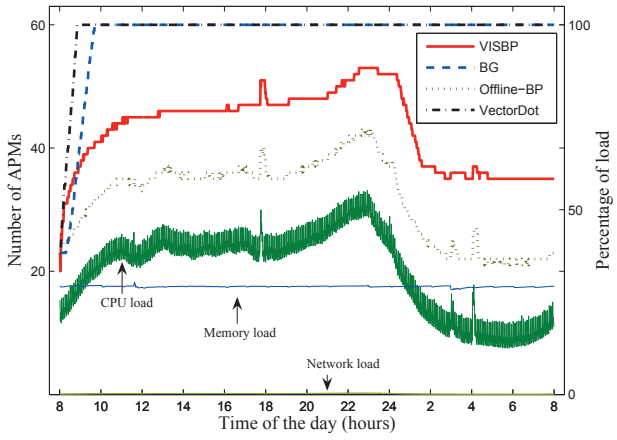


Fig. 2. Active PMs with the daily load variation

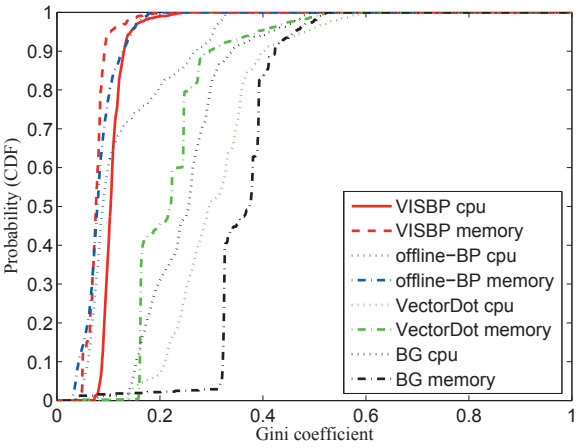


Fig. 4. Gini index of CPU and memory

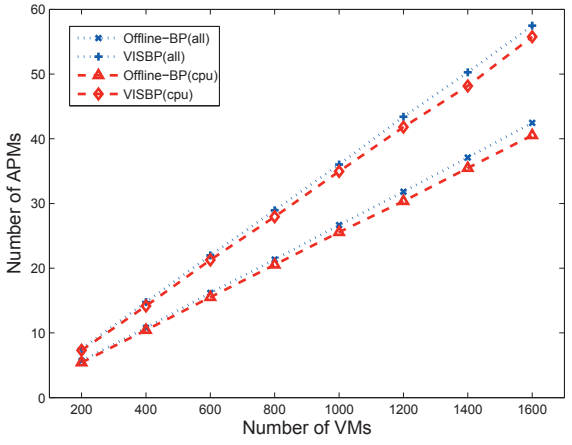


Fig. 3. APM comparison of offline-BP and VISBP

indicates that bin packing based algorithms use much fewer APMs than the other two algorithms. Recall that both the BG and the VectorDot algorithms have no support for green computing. Hence, they quickly use up all PMs and never release them. In other words, once the VMs are spread out during high load, they are never consolidated when the load goes down. In contrast, the resource usage of the two bin packing algorithms closely mimic the variation of the load in the system, indicating good green computing effect. The figure also shows that our algorithm uses more APMs than the offline algorithm. This is investigated in more details in Figure 3 where we compare the average number of APMs in the two algorithms when the scale of the system increases.

In figure 3 we compare the average numbers of APMs in offline bin packing and our online version of the algorithm. We found that our algorithm uses about 27% more PMs than the offline algorithm. This percentage is independent of the scale and the number of resource dimensions. As we will see in Section 4.5, offline bin packing algorithms incur too many VM migrations and are not suitable in a cloud computing environment.

4.4 Load balance

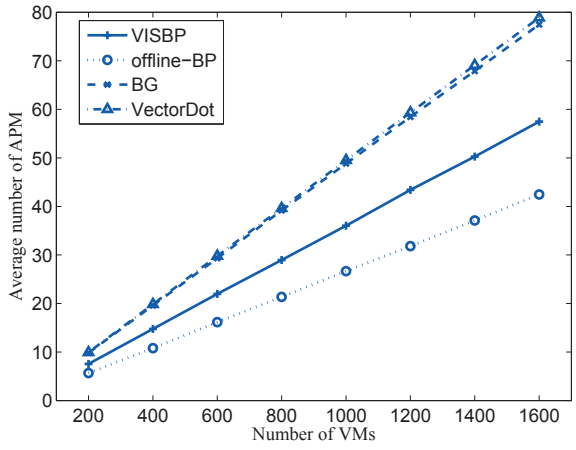
We use the Gini coefficient [28] to measure the distribution of resource utilizations across APMs. Gini coefficient is widely used to measure the distribution of wealth in society. It is calculated from the normalized area between the observed CDF and the CDF of an uniform distribution. A smaller coefficient indicates a more equal distribution. Figure 4 shows the CDF of Gini coefficient for the CPU and the memory resources. As can be seen from the figure, our online algorithm maintains an even load distribution similar to the offline version, and much better than the BG and the VectorDot algorithms.

4.5 Scalability

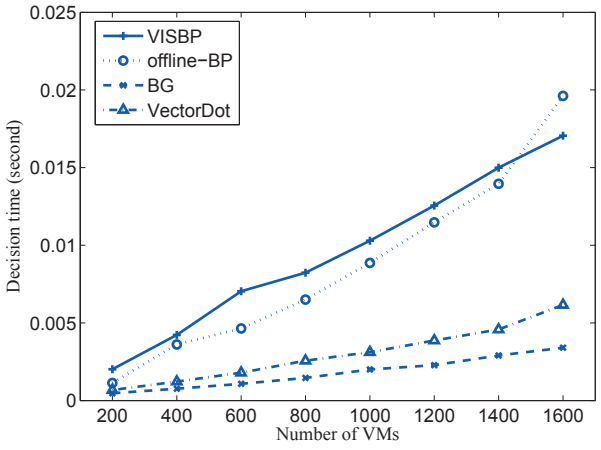
We evaluate the scalability of the algorithms by increasing the number of VMs in the simulation from 200 to 1600 while keeping the VM to PM ratio fixed at 20:1 as said before. Figure 5 (a) shows that, for all four algorithms, the average number of APMs increases linearly with the system size. Our VISBP algorithm uses about 73% of total PMs, much fewer than 98% of the BG algorithm and 99% of the VectorDot algorithm. This demonstrates that our VISBP algorithm achieves good green computing effect. Although the offline BP algorithm uses 19% fewer PMs than ours, as shown later in this section, it incurs too many migrations to be practical.

Figure 5 (b) shows that the average decision time of all algorithms increase roughly linearly with the system size. Although the VISBP algorithm seems slower than the BG and the VectorDot algorithms, the decision time is a trifle compared with the scheduling interval. When the size reaches 1600, the average decision time is still under 0.02 seconds per scheduling run.

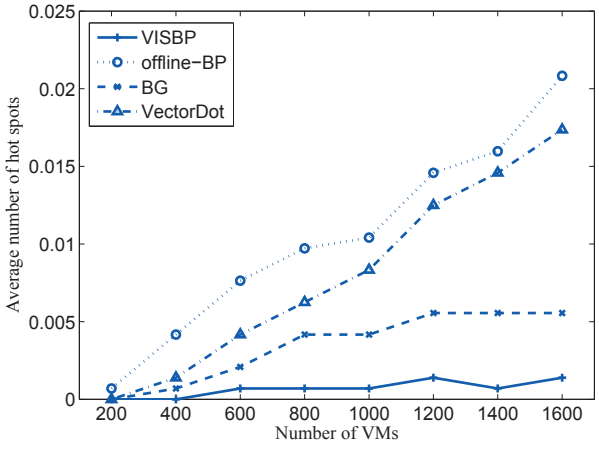
Figure 5 (c) compares the average number of hot spots in the system. Recall that a PM is considered a hot spot when its utilization is above a predefined hot threshold. Here we use a 90% hot threshold. The numbers of hot spots for all four algorithms increase roughly linearly with the system size. With the VISBP algorithm, the average possibility



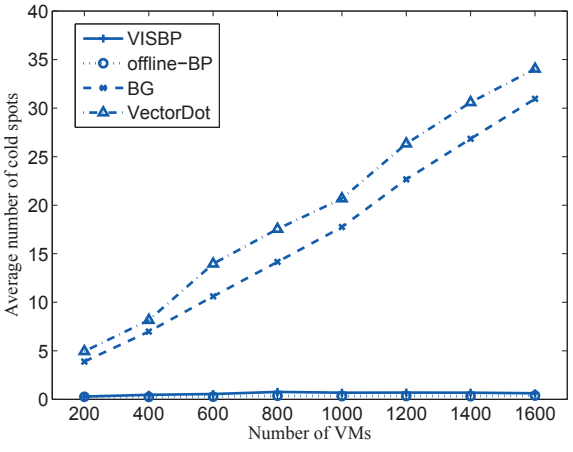
(a) number of APMs



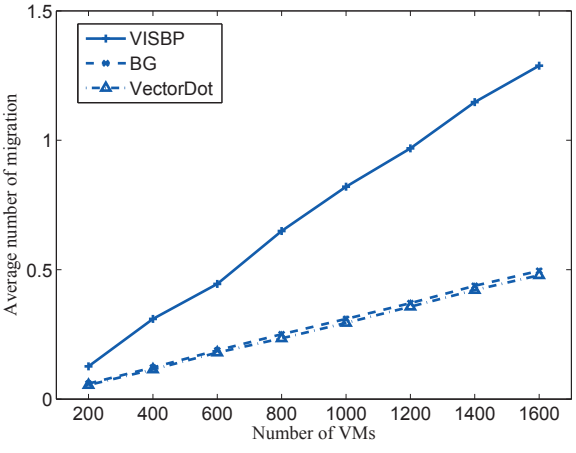
(b) average decision time



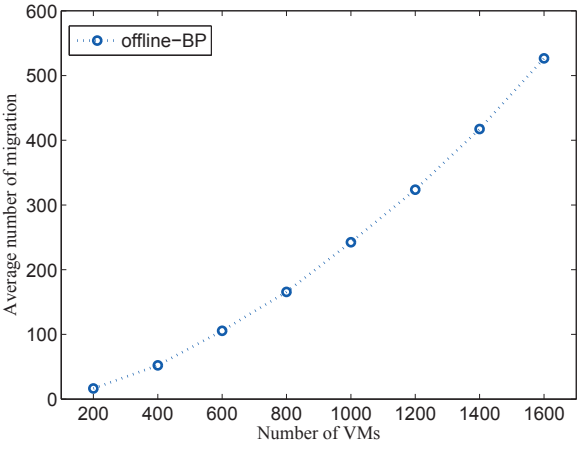
(c) number of hot spots



(d) number of cold spots



(e) number of migrations



(f) number of migrations

Fig. 5. Scalability of the algorithms with system size

of being a hot spot for a PM in each decision interval is 1.75×10^{-5}. It outperforms the other three algorithms. The VISBP algorithm uses only unfilled or unused bins to receive an item even though some closed bins might have enough space for that item. This feature makes it likely that the bins have some spare space for absorbing fluctuation of item sizes. The offline bin packing algorithm does not enjoy this feature. Although the VectorDot and the BG algorithms try best to scatter the VMs, they are driven by the shortage of resource. In contrast, the VISBP algorithm adjusts the VM layout in advance.

We define a cold spot as a PM whose utilization of

all resources is under 25%[4]. The number of cold spots represents the extent of resource waste. Figure 5 (d) shows that the VISBP incurs much fewer cold spots than the BG and the VectorDot algorithms.

We also calculate the total number of migrations at varying scale. As shown in Figure 5 (e) and (f), for all algorithms, the number of migrations increases roughly linearly with the system size. In our VISBP algorithm, each VM on average experiences 8×10^{-4} migrations during each decision run, independent of the system size. This translates into roughly one migration per thousand decision intervals. The stability of our algorithm is good enough. Figure 5 (f) shows that the number of migrations of offline-BP algorithm is much more than that of ours. With the scale of 1600 VMs, a VM experiences one migration every three decision intervals by the offline-BP algorithm, and a PM experiences seven migrations every decision interval. This incurs high network overhead and is hard to finish in each decision interval.

4.6 Determine the α parameters for load predictor

For both $\uparrow \alpha$ and $\downarrow \alpha$, we have tested 21 values from -1.0 to $+1.0$ with step length 0.1 by simulation. All 441 combinations are tested in a system with 1000 VMs and 50 PMs to study their effects on the performance of VISBP. We find that when $\uparrow \alpha$ is in $[-0.4, 0.3]$ and $\downarrow \alpha$ in $[0, 0.9]$, the number of hot spots is fewer than 0.0007 per decision run. Once $\uparrow \alpha$ grows above 0.8, the number of hot spots grows rapidly above 0.0035 per decision run. This is because the predictor fails to identify the rising resource demand and leads to under-provision. A similar situation occurs with negative $\downarrow \alpha$.

The number of migrations grows when both α values drop. If both α values drop in to $[-0.5, 0.8]$, the number of migrations grows nearly linearly from 0.77 to 0.87 per decision run. The number of migrations grows radically with α values drop below -0.5. Since the negative α magnifies the fluctuation of load, the algorithm increases the number of migration to accommodate the change of resource demands. If α values increase over 0.8, load fluctuation is smoothed so that the number of migration decreases. However, this can lead to under-provisioning if the load increases really fast.

The number of cold spots and the number of APMs grow slowly and linearly when $\downarrow \alpha$ increases or $\uparrow \alpha$ declines. Larger $\downarrow \alpha$ prevents the predicted load from dropping too fast while lower $\uparrow \alpha$ makes it increase faster. This leads to more resource reservation to mitigate potential hot spots. The result showes that, with $\uparrow \alpha = -0.9$ and $\downarrow \alpha = 0.9$, the algorithm uses only one more APM than the case without the predictor.

Since the two α values work well in a wide range, we have adopted $\uparrow \alpha = -0.2$ and $\downarrow \alpha = 0.7$ in both the simulation and the experiments, which are around the center in that range.

4. It has been shown in the literature that the average resource utilization of typical data centers is between 20% and 30% [7]. So we use 25% (the middle of 20% and 30%) utilization as threshold for under-utilized servers.

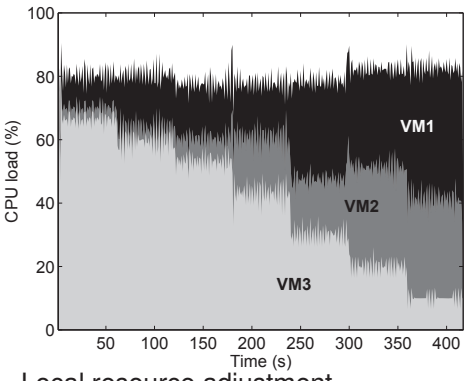


Fig. 6. Local resource adjustment

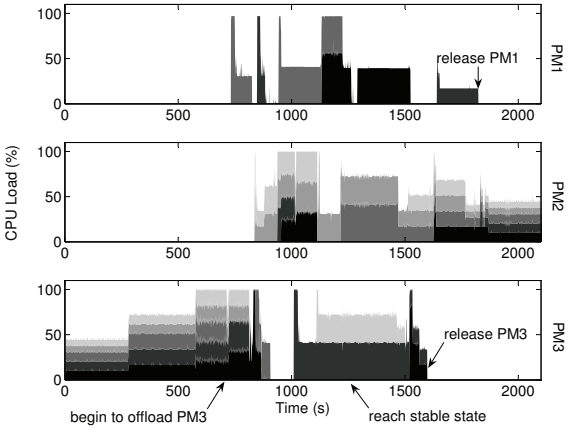


Fig. 7. One dimensional resource experiment

5 Experiments

The experiments are conducted using a group of servers that run the Xen hypervisor. The servers are connected to a shared NFS storage server providing storage for VM images. Our algorithm is implemented as an Usher plug-in.

5.1 Local resource adjustment

Our first experiment demonstrates the effectiveness of local resource adjustment. We run three VMs on a server. The server has one Intel E5420 core and eight gigabyte of RAM. An Apache web server runs in each VM. We use httperf to invoke CPU intensive PHP scripts on Apache servers. The total CPU load of the three VMs is kept constant at 80%, but the load of each VM varies with time. The default credit scheduler in Xen is used. Figure 6 shows that local resource adjustment can adapt to the changing demand well.

5.2 One dimensional resource experiment

Now we evaluate the effectiveness of the VISBP algorithm in overload mitigation and green computing. We use three physical servers (PMs) and five VMs in this experiment. The configuration of the PMs conforms to the one used in Section 5.1. All VMs are configured with 128 MB of RAM and are on PM3 initially. Figure 7 shows the VISBP algorithm in action with the changing load. Different shades are used for each VM. We first increase the CPU load of the

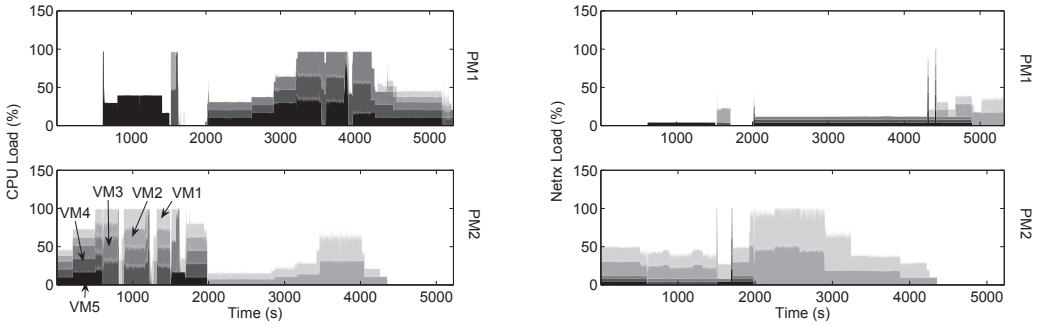


Fig. 8. Multi-dimensional resource experiment

VMs gradually to create an overload on PM3. The figure shows that the VISBP algorithm starts a series of migrations to offload PM3, first to PM1, and then to PM2. It reaches a stable state under high load around 1250 seconds. Then we decrease the CPU load of all VMs gradually. Now green computing takes effect. Around 1600 seconds, two VMs were migrated from PM3 to PM2 so that PM3 can be put into the standby mode. Around 1800 seconds, a VM is migrated away so that PM1 can be released. As the load goes up and down, the VISBP algorithm will repeat the above process: spread over or consolidate the VMs as needed.

5.3 Multi-dimensional experiments

Now we see how the VISBP algorithm handles a mix of CPU and network intensive workloads. We vary the CPU load as before. We inject the network load by sending the VMs a series of network packets. The results are shown in Figure 8. We reduce the number of PMs to two so that they can fit into the figure easily. The two rows represent the two PMs. The two columns represent the CPU and the network dimensions, respectively. The first part of the figure is similar to the previous one. Then starting at time 2000 seconds, we increase the network load of VM1 and VM2 from 20% to almost 50%. This causes VM3, VM4, and VM5 to move to PM1. Note that VM1 and VM2 are not migrated. PM1 and PM2 correspond to the SS-bin and the T-bin in our bin packing algorithm, respectively. The subsequent increase in CPU load does not break the properties of these two bins. From 4000 seconds, the CPU and network load of all VMs decline, which eventually makes all VMs T-items. Around 4400 seconds, VM1 and VM2 are migrated to PM1 so that PM2 is released. This experiment demonstrates that the VISBP algorithm works equally well for multi-dimensional resources.

5.4 Comparison of algorithms

We setup a prototype data center environment to compare our algorithm along with the others. We use 10 servers as computing nodes to run 80 virtual machines. The scheduler runs on a dedicated management server. Two NFS servers provide a centralized storage for VM images. Each server reserves 10 gigabytes of memory and four

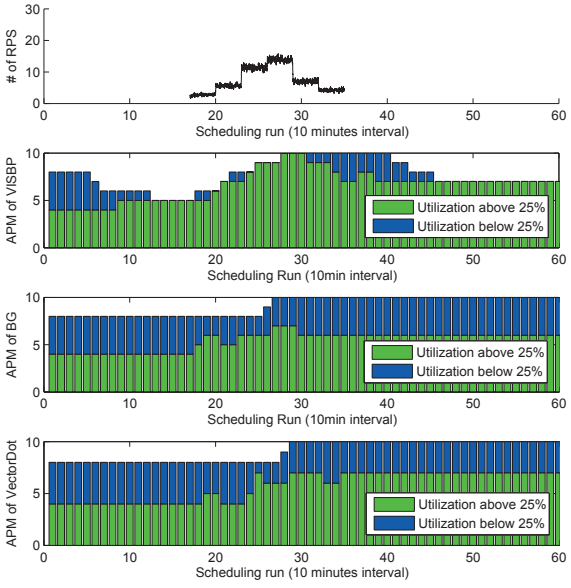


Fig. 9. Comparison of algorithms

CPU cores for VMs. Each VM is configured with two gigabyte initial memory and one Virtual CPU. We disable the Hyper-Threading function so that each running virtual CPU can get a full core. Self-ballooning is enabled inside each VM to reclaim unused memory.

We use the TPC-W [29] benchmark to simulate real world workload. The TPC-W benchmark is composed of the server side and the client side. The server side implements a web application of an online bookstore. The client side imitates the browsing and the buying behaviors of the customers. In our experiment, Each VM runs a copy of the server side. To simulate rising and declining of the load, we prepare six consecutive waves of hybrid workload including both the browsing and the ordering behaviors at different throughput levels.

The initial layout of the 80 VMs is fixed. Four of the servers run 15 VMs each, another four run seven, six, four, and three VMs respectively. The other two servers are idle. The scheduling algorithm is invoked every ten minutes. The predictor parameters conform to those used

in the simulation. Workload is started synchronically by a script.

Figure 9 shows the scheduling process of the VISBP algorithm. The first figure shows the workload fed to each VM. The number of request per second (RPS) rises gradually to 15; then it drops down step by step. In the second figure, we see that VISBP consolidates the VMs to five servers in the thirteenth scheduling interval (or 130 minutes). Half of the servers are idle. They can be put to standby mode to save power and waken up on demand by technologies such as Wake-on-LAN. After the load passes, the VISBP algorithm again consolidates VMs distributed on ten servers to seven servers. It uses two more servers than the previous consolidation process because the operating systems in the VMs use more memory to cache data during the workload. In contrast, as shown in the last two figures in Figure 9, the BG and the VectorDot algorithms do not consolidate VMs. As a result, they generate many under-utilized servers.

We have also observed that the VISBP algorithm uses more migrations than the BG and the VectorDot algorithms. It on average incurs 1.03 migrations in each scheduling interval. The maximum number of migrations it has in a single scheduling round is 5. Even in this worst case, all migrations can be completed in less than 90 seconds. As shown in the simulation, migration tends to be much less frequent in the real environment because real workload is generally more stable than our synthetic workload.

Prudent readers may find from figure 9 that the resource utilization of idle VMs is not low. An idle VM can use about 3% of the resource capacity of a server. This is because about 300 megabytes of memory is consumed once the application is started. However, the demand on memory does not increase much when the workload increases to peak load. In contrast, the CPU usage grows linearly to 8% of server capacity and become the bottleneck resource, leaving space for about 12 VMs per host.

Interestingly, throughout this experiment, none of the algorithms let the resource utilization of a server rising above 90%. This demonstrates the effectiveness of our load predictor.

We also conduct this experiment using the offline bin packing algorithm, but it requests so many migrations that the Xen hypervisors in some servers stop responding. It cannot finish a complete procedure even though we have tried many times.

6 Related Work

6.1 Resource Allocation

The problem of adaptive resource allocation in the cloud environment has been investigated in the literature. To achieve application level quality of service (QoS), Padala [30] [31] models application performance as a linear function of allocations of CPU time and disk I/O so that the resource allocator on each physical server can smartly adjust the allocation to get expected performance. In Seawall [32], a centralized controller forecasts congestion in data center network from traffic among application components. It carefully throttles the network I/O of involved virtual machines to optimize network resource utilization. The above approaches focus on fairness in resource allocation across the applications. Our work, in contrast, focuses on hot spot avoidance and green computing.

MSRP replicates web applications to meet their resource requirement [4]. It allows web applications to bid for server resources. A resource allocation algorithm decides which applications are served to maximize profit by minimizing the number of profitable active servers. Hot spots are avoided since the algorithm prohibits over-commitment. However, this algorithm assumes that the memory footprint of applications is small enough that a single server can host all applications in a data center, which is no more applicable nowadays. In modern systems, memory is treated as the bottleneck resource [8] [33]. A server can only host a limited number of applications and the replacement of applications is an expensive operation. As such, they have a strong motivation to minimize application placement changes. In our previous work, we use a CCBP model to provide automatic scaling for web applications in the cloud environment. It aims at good demand satisfaction ratio, minimal replacement, and power consumption. These systems do not use virtualization and apply only to specific applications. In contrast, we treat each VM as a black box and have no knowledge of encapsulated applications.

MapReduce [34] is also a popular framework that supports the resource allocation in the cluster. However, it often runs in higher level than our system. A lot of research has run MapReduce as an application in the cloud, e.g. Amazon EC2 and MCP [35]. Therefore, our VISBP approach can still be used to adjust the resource allocation effectively in these cases.

Similar to our work, Sandpiper [9] and HARMONY [11] exploit live migration to offload hot spots. However, they do not support green computing. Some approach invokes classical offline bin packing algorithm periodically to optimize resource allocation [10]. Although the offline algorithm does well in green computing, the number of migrations it incurs is practically prohibitive. Interestingly, Rahman et al. tries to prevent unnecessary migration by solving contention locally [36].

6.2 Green Computing

There are many approaches to green computing in data centers. One category leverages hardware technologies to improve power efficiency of each server. Another adopts low power agencies to minimize idle power consumption.

Close to our approach, a bunch of resource scheduling systems try to minimize the number of active physical servers [4] [10] [37]. Idle servers may be put into sleep state or even shutdown to save power. This approach is known as server consolidation. As shown in VirtualPower, server consolidation accounts for the majority of power saving effect [18]. In addition, this approach does not rely on any

special hardware or specific settings like *Sleep Server* [6]. VMAP incorporates the idea that resource utilization of servers in better cooled areas of a data center can be higher than that of the others for better power efficiency [38]. This idea can be adopted in our future work.

6.3 Bin packing algorithms

In approach given in [10], the schedulor periodically invokes offline bin packing algorithm to calculate the mapping from VMs to PMs. Stillman uses bin packing algorithm in a job system to optimize the job execution time [39]. Wang et al. incorporate probability analysis into the classical offline First-Fit bin packing algorithm as a tool for network resource allocation in data centers [12]. The above approaches all adopt offline bin packing algorithms. Balogh and Bëkesi investigate various online bin packing algorithms and proves a new lower bound on the approximation ratio [40]. Our algorithm is derived from Gambosi's algorithm to take advantage of its control over adjustment cost [21].

Theoretical analysis on the average performance of our online bin packing algorithm is a complicated task depending on the probability distribution of the item sizes (i.e., the VM load). Existing work has some intriguing results for the uniform distribution. They use the average *waste space*, the aggregate unused space in open bins, as an indicator for the average performance of a bin packing algorithm. For example, analysis has shown that, for one-dimensional first-fit decreasing (FFD) bin packing algorithm[5], the waste space is $O(\sqrt{n})$ if item size conforms to uniform distribution over $(0, 1]$, where n is the number of items to be packed [41] [42]. If item size conforms to uniform distribution over $(0, a]$, where $0 < a < 1$, the wasted space is $O(1)$ for $a \leq 1/2$ and $O(\sqrt[3]{n})$ for $1/2 < a < 1$ [43]. Karp et al. also conduct analysis on VPACK, a special vector bin packing algorithm where the size in each dimension of an item is independent and conforms to an uniform distribution over $[0, 1]$ [44].

7 Conclusion

We have presented the design, implementation, and evaluation of a practical online bin packing algorithm called the VISBP which can allocate data center resources dynamically through live VM migration. Compared to the existing algorithm, the VISBP excels in hot spots mitigation and load balance. It also reconciles the demands on both green computing and stability.

In the future, we plan to incorporate application performance model and allocating resource according to service level agreement. In addition, we plan to leverage the memory similarity among virtual machines with memory de-duplication technologies to improve the ratio of VM to PM as well as the performance of live migration.

5. offline-BP degrades to FFD when only one resource dimension is considered

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61170056), the National High Technology Research and Development Program (863 Program) of China (Grant No. 2013AA013203), National Basic Research Program of China (Grant No. 2009CB320505) and Digital Resource Security Protection Service Based on Trusted Identity Federation and Cloud Computation SubProject of 2011 Information Security Special Project sponsored by National Development and Reform Commission.

References

[1] G. Rosen. (2010, Dec.) Recounting ec2 one year later. [Online]. Available: http://www.jackofallclouds.com/2010/12/recounting-ec2/

[2] unknown. (2012, Mar.) Amazon data center size. [Online]. Available: http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*, 2005.

[4] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proc. of the 18th ACM Symposium on Operating System Principles (SOSP'01)*, 2001.

[5] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: augmenting network interfaces to reduce PC energy usage," in *Proc. of the 6th USENIX symposium on Networked systems design and implementation (NSDI'09)*, 2009.

[6] Y. Agarwal, S. Savage, and R. Gupta, "Sleepserver: a software-only approach for reducing the energy consumption of PCs within enterprise environments," in *Proceedings of the USENIX conference on USENIX annual technical conference*, 2010.

[7] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *Proc. of the 14th international conference on Architectural support for programming languages and operating systems (ASPLOS'09)*, 2009.

[8] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proc. of the 16th International World Wide Web Conference (WWW'07)*, 2007.

[9] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and Gray-box strategies for virtual machine migration," in *Proc. of the 4th Symposium on Networked Systems Design and Implementation (NSDI'07)*, 2007.

[10] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Proc. of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07)*, 2007.

[11] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *Proc. of the 20th ACM/IEEE conference on Supercomputing (SC'08)*, 2008.

[12] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. of 30th IEEE International Conference on Computer Communications (INFOCOM'11)*, 2011.

[13] H. Xu and B. Li, "Egalitarian stable matching for VM migration in cloud computing," in *Proc. of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2011.

[14] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: coordinated multi-level power management for the data center," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, 2008.

[15] S.-H. Lim, S.-J. Huh, Y. Kim, and C. R. Das, "Migration, assignment, and scheduling of jobs in virtualized environment," in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing*, 2011.

[16] J. Zhu, Z. Jiang, Z. Xiao, and X. Li, "Optimizing the performance of virtual machine synchronization for fault tolerance," *IEEE Transactions on Computers (TC)*, December 2011.

[17] J. Zhu, Z. Jiang, and Z. Xiao, "Twinkle: A fast resource provisioning mechanism for internet services," in *Proc. of IEEE Infocom*, April 2011.

[18] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *Proceedings of 21st ACM SIGOPS symposium on Operating systems principles (SOSP'07)*, 2007.

[19] M. R. Garey and D. S. Johnson, "A 71/60 theorem for bin packing," *Journal of Complexity*, vol. 1, 1985.

[20] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *Journal of the ACM (JACM)*, vol. 32, no. 3, 1985.

[21] G. Gambosi, A. Postiglione, and M. Talamo, "Algorithms for the relaxed online bin-packing model," *SIAM J. Comput. Issue 5*, vol. 30, 2000.

[22] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Dynamic bin packing," *SIAM Journal on Computing*, vol. 12, 1983.

[23] J. W.-T. Chan, T.-W. Lam, and P. W. Wong, "Dynamic bin packing of unit fractions items," *Theoretical Computer Science*, vol. 409, 2008.

[24] L. Epstein and M. Levy, "Dynamic multi-dimensional bin packing," *Journal of Discrete Algorithms*, vol. 8, 2010.

[25] C. Chekuri and S. Khanna, "On multi-dimensional packing problems," in *Proc. of the ACM-SIAM symposium on Discrete algorithms*, 1999.

[26] G. Galambos and G. J. Woeginger, "On-line bin packing-a restricted survey," *Physica Verlag*, vol. 42, no. 1, 1995.

[27] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: An extensible framework for managing clusters of virtual machines," in *Proc. of the 21st Large Installation System Administration Conference (LISA'07)*, 2007.

[28] S. Yitzhaki, "Relative deprivation and the gini coefficient," *The Quarterly Journal of Economics*, vol. 93, may 1979.

[29] "TPC-W: Transaction processing performance council, http://www.tpc.org/tpcw/."

[30] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *Proc. of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.

[31] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. of the 4th ACM European conference on Computer systems (EuroSys'09)*, 2009.

[32] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: performance isolation for cloud datacenter networks," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.

[33] Z. Xiao, Q. Chen, and H. Luo, "Automatic scaling of internet applications for cloud computing services," *to appear in IEEE Transactions on Computers*, 2013.

[34] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, 2008.

[35] Q. Chen, C. Liu, and Z. Xiao, "Improving mapreduce performance using smart speculative execution strategy," *to appear in IEEE Transactions on Computers*, 2013.

[36] M. M. Rahman, R. Thulasiram, and P. Graham, "Differential time-shared virtual machine multiplexing for handling qos variation in clouds," in *Proceedings of the 1st ACM multimedia international workshop on Cloud-based multimedia applications and services for e-health*, 2012.

[37] G. Chen, H. Wenbo, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008.

[38] E. K. Lee, H. Viswanathan, and D. Pompili, "Vmap: Proactive thermal-aware virtual machine allocation in hpc cloud datacenters," in *Proceeding of IEEE Conference on High-Performance Computing (HiPC'12)*, 2012.

[39] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource allocation using virtual clusters," in *Proc. in the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'09)*, 2009.

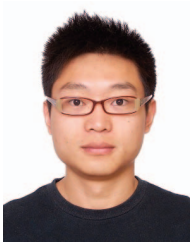
[40] J. Balogh and J. Békési, "Semi-on-line bin packing: a short overview and a new lower bound," *Central European Journal of Operations Research*, 2012.

[41] W. Knödel, "A bin packing algorithm with complexity o(n log n) and performance 1 in the stochastic limit," in *Proceedings on Mathematical Foundations of Computer Science*, 1981.

[42] R. Loulou, "Probabilistic behaviour of optical bin-packing solutions," *Operations Research Letters*, vol. 3, no. 3, 1984.

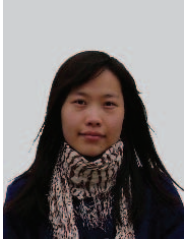
[43] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch, "Some unexpected expected behavior results for bin packing," in *Proceedings of the 16th annual ACM symposium on Theory of computing*, 1984.

[44] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela, "A probabilistic analysis of multidimensional bin packing problems," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984.



Weijia Song received bachelor's degree and master's degree from Beijing Institute of Technology. He is currently a doctoral student at Peking University. His current research focuses on resource scheduling problems in Cloud systems.

Zhen Xiao is a Professor in the Department of Computer Science at Peking University. He received his Ph.D. from Cornell University in January 2001. After that he worked as a senior technical staff member at AT&T Labs - New Jersey and then a Research Staff Member at IBM T. J. Watson Research Center. His research interests include Cloud Computing, virtualization, and various distributed systems issues. He is a senior member of ACM and IEEE.



Qi Chen Qi Chen received bachelor's degree from Peking University in 2010. She is currently a doctoral student at Peking University. Her current research focuses on the Cloud Computing and Parallel Computing.

Haipeng Luo received bachelor's degree from Peking University in 2011. He is currently a doctoral student at Princeton University. His current research focuses on machine learning.