

Hierarchical Virtual Machine Consolidation in a Cloud Computing System

Inkwon Hwang and Massoud Pedram

University of Southern California
Los Angeles, CA, USA
{inkwonhw, pedram}@usc.edu

Abstract — Improving the energy efficiency of cloud computing systems has become an important issue because the electric energy bill for 24/7 operation of these systems can be quite large. The focus of this paper is on the virtual machine (VM) consolidation in a cloud computing system as a way of lowering daily energy consumption of the system. In contrast to the existing works that assume resource demands of VMs are known and given as scalar variables, this paper treats these demands as random variables with known means and standard deviations. These random variables may be correlated with one another, and there are several kinds of resources which can be performance bottlenecks. Therefore, both the correlation and multiple resource type should be considered. The VM consolidation problem is then formulated as a multi-capacity stochastic bin packing problem. This problem is NP-hard, so we propose a heuristic method to solve the problem efficiently. The simulation results show that, in spite of its simplicity and scalability, the proposed method produces high quality solutions.

Keywords- Cloud computing; portfolio effect; multi-capacity bin packing; stochastic; virtual machine

I. INTRODUCTION

Data center's energy efficiency was not a major concern as of only a few years ago. However, the electric energy bill for operating a typical data center has been increasing rapidly as the size and utilization level of the data center increases; therefore, improving the energy efficiency has become a critically important issue.

The cloud computing system considered here consists of one or more (federated) data centers. The data center itself is composed of a large number (say, tens of thousands) of heterogeneous server machines. Because each server machine consumes hundreds of Watts, the gross power consumption of these server machines plus the air conditioning units in a typical data center can easily exceed a few MW. For example, one of Facebook's data centers in Prineville, Oregon has a capacity for 28MW of power [1]. The power capacity of very large data centers comes close to 100 MW of power, which is the same as the power consumed by 80,000 U.S. homes or 250,000 E.U. homes [2]. With 10-12 cents per KWhr of electrical energy consumed, the electrical energy bill for even a mid-size data center (say with 10MW average power consumption) exceeds twenty thousand dollars per day. Because of this huge cost, there is a growing need for energy-

aware resource management strategies in data centers. Considering that a typical data center is under-utilized much of the time, the energy cost can be greatly reduced by consolidating service requests and/or running applications into as few servers as possible and shutting down the surplus servers. This technique is known as *server consolidation*. This consolidation enhances the energy efficiency of data centers because of the non-energy proportional characteristics of the real servers [3]. Note that in a virtualized cloud system, the key consolidation task is to pack the existing virtual machines (VM) into the minimum number of server machines. This server consolidation for a virtualized system is called *VM consolidation*.

Much of the existing work proposes deterministic methods of consolidation, assuming that the resource demands are known precisely and given as scalar values [4]. This assumption, however, is generally invalid. First, the estimated resource demands must represent actual demands during relatively long periods of time, which range from a few minutes to a few hours [5]. It is because the *VM migration*, which is an essential part of the consolidation, cannot be done too frequently due to the migration overhead; in fact, even the live migration, which is a very efficient migration method, causes service down for a few hundred milliseconds [5]. In practice, the actual demands vary during such a long period of time. If the VM resource demands are modeled as scalar values, then they must be set to very large values in order to account for the worst case. Clearly, this is unnecessary and wasteful. Second, some types of VM resource demands are very bursty with rates that vary greatly and rapidly over time; e.g., consider the problem of packing multiple connections together on a link when each connection is bursty [6]. In such a case, it has proven useful to rely on the notion of an effective bandwidth for bursty connections and to model this bandwidth as a random variable with expected mean and standard deviation (which are estimated by dynamic profiling of the connection). Therefore, it can be inappropriate to characterize the VM resource demands by fixed values. Instead, we suggest characterizing the demands by random variables (RV).

There are prior studies which treat resource demands as random variables [7-11]; these works formulate the energy aware resource allocation problem as the *stochastic bin packing* (SBP) problem, which states that items with the sizes following a probabilistic distribution must be packed into bins such that the minimum number of bins are used, and the probability of violating any bin size is below a given threshold. Meng *et al.* [7], who focus on the network bandwidth as the resource in question, make a couple of assumptions: 1) items

This research is sponsored in part by a grant from the Semiconductor Research Corporation (No. 2012-HJ-2292).

(VMs with known network bandwidth demands) are independent RVs following a normal distribution, 2) bins (physical machines, PM, with known network capacities) are identical. The authors subsequently present an algorithm to solve the SBP based on the notion of the *equivalent size* of an item, which in general depends on the other items packed together in the same bin. Similarly, Breitgand and Epstein [11] consider consolidating VMs on the minimum number of PM where the physical network (e.g., network interface link) may become the bottleneck. In their formulation, each VM has a probabilistic guarantee (derived from its *Service Level Agreement*, SLA) on realizing its bandwidth requirements. The problem is again set up as SBP problem where each VM's bandwidth demand is treated as an RV following a normal distribution. The authors also assume the RVs are independent of one another. On the other hand, Ming *et al.* and Xiaoqiao *et al.* do not assume neither the identical bin size nor the independency among RVs, but only one resource type (i.e., the CPU) is considered [8, 10].

These assumptions, however, are generally invalid. First, VMs can be strongly correlated; therefore, their resource demands may be dependent on one another. For example, a web service request (such as a Google query) may simultaneously be served by a number of VMs (i.e., load balancing), resulting in correlation among the resource demands of these VMs. Second, it is important to consider multiple resource types (e.g., CPU, network bandwidth, disk space, and memory size) since any one of them may become the performance bottleneck. Third, a typical data center consists of heterogeneous PMs in general; hence, it is inappropriate to assume all PMs are identical (the same bin size). Without these invalid assumptions, we formulate the consolidation problem into a *multi-capacity stochastic bin packing* (MCSBP) problem¹ and propose a heuristic method to solve the problem. This method is hierarchical and thus highly scalable.

The remainder of the paper is organized as follows. In Section II, we introduce the resource demand model, the concept of portfolio effect, and the problem statement. The proposed algorithms and detailed explanations about them are presented in Section III. In Section IV, the simulation results are presented. Finally, in Section V, we summarize and conclude.

II. RESOURCE DEMAND MODEL AND PROBLEM STATEMENT

A. Resource demand model

This study assumes that the VM's resource demands are specified as RVs. The demands depend on the characteristics and computing needs of the applications running on the VMs. If the *cumulative distribution function* (*cdf*) of a RV is known, the minimum amount of resource allocation can be estimated from this *cdf* to meet a target *quality of service* (QoS). This QoS may be specified as the probability that the aggregate resource demand of VMs in a PM does not exceed the PM's resource capacity by more than 5% (this is referred to as having a 95% QoS). The *cdf* of RVs, however, is unknown in many cases. Without knowledge of the *cdf*, the minimum

amount of resource allocation can be estimated by the *Cantelli's inequality* [12], which is the single-tailed variant of *Chebyshev's inequality*.

$$P\{X \geq \mu_x + \beta\sigma_x\} \leq \frac{1}{1 + \beta^2}, \beta > 0 \quad (1)$$

According to the *Cantelli's inequality*, the minimum amount of resource needed to meet the target QoS can be estimated as:

$$\mu_x + \beta\sigma_x, \beta = 4.4 \text{ for } 95\% \text{ QoS target} \quad (2)$$

This inequality holds for any RVs regardless of their distribution, so it does not give a tight bound and may cause resource overbooking. If more information about the VM is given (e.g., *cdf*), we can assign less amount of resource while meeting the same QoS. For example, if a RV is known to be normally distributed, β can be set to as low as 1.7, which is much smaller than what the *Cantelli's inequality* gives. According to the *central limit theorem* (CLT), the mean of a sufficiently large number of independent RVs, each with finite mean and variance, approximately follows normal distribution [12]. The CLT holds even for weakly dependent RVs; hence, we can use smaller β (i.e., a value of 1.7) if a RV is the sum of a large number of weakly dependent RVs.

This study takes care of multiple resource types, e.g., the CPU, memory, network bandwidth, and so on. If the demand for each resource type is modeled as a RV, there are too many RVs, which make the problem complicated and hard to solve. One difficulty is that a correlation coefficient matrix becomes too big, which leads to large memory usage as well as longer running time to solve the problem. Instead, the workload intensity of a VM is modeled as a RV, and a resource demand is defined as a linear function of the RV X_n (3):

$$R_n^1 = a_n^1 X_n + b_n^1, \dots, R_n^k = a_n^k X_n + b_n^k \quad (3)$$

where X_n is a RV modeling the workload intensity of the n^{th} VM (VM_n), R_n^k is the demand for the k^{th} resource type, and a_n^k and b_n^k are regression coefficients. This linear model is reasonable because there is a correlation between the workload intensity and the resource demands. If a resource type is very weakly correlated with the workload intensity, e.g., the memory, a relatively small value of a_n^k can be chosen for the resource.

B. Portfolio effect

The *modern portfolio theory* (MPT) is a financial theory, which attempts to maximize a portfolio's expected return for a given amount of portfolio risk, or equivalently minimize the risk for a given level of expected return, by carefully choosing the proportions of various assets. The MPT models an asset's return as a RV, and defines risk as the standard deviation of the return. The MPT models a portfolio as a weighted combination of assets, so that the return of the portfolio is the weighted combination of the assets' returns. By combining different assets whose returns are not perfectly positively correlated, the MPT seeks to reduce the total risk (variance of the portfolio return) [13].

The MPT reduces risk of portfolio through the *portfolio effect*, which may be stated as follow; the risk of a portfolio is always less than or equal to sum of each asset's risk (4).

$$\mu_Y = \sum_i \mu_{X_i}, \sigma_Y^2 = \sum_i \sum_j \rho_{ij} \sigma_i \sigma_j \leq \left(\sum_i \sigma_i\right)^2 \quad (4)$$

where $Y = \sum_i X_i$ and ρ_{ij} is a correlation coefficient between X_i and X_j ($-1 \leq \rho_{ij} \leq 1$).

¹ Note the *multi-capacity bin packing* problem differs from the *multi-dimensional bin packing* problem. Detailed comparison is shown in Section III.B.

The degree of risk reduction is a function of a correlation coefficient (ρ_{ij})—the smaller ρ_{ij} is, the lower the risk is (cf. Figure 1). In other words, one has to avoid from putting highly positively correlated assets into the same portfolio.

As shown in (2), the minimum resource allocation is proportional to both mean and standard deviation; hence, reduction in the standard deviation also reduces the amount of resource allocation. The proposed method maximizes this *portfolio effect* to minimize the resource allocation, and consequentially energy cost decreases by utilizing fewer PMs.

C. Problem statement – multi-capacity stochastic bin packing optimization problem

Assume there are M PMs, N VMs, and K resource types. Each PM has a set of deterministic capacity limits for each resource type, and the limit is specified as r_m^k for the k^{th} resource capacity of the m^{th} PM (PM _{m}). The ON/OFF state of PM _{m} is captured by a pseudo-Boolean variable f_m . In addition, a non-deterministic workload intensity of VM _{n} is specified as a X_n (a RV with known mean μ_n and variance σ_n^2)². The demands by a VM for each type of resource are given as linear equations of the VM's workload intensity (3). Let ρ_{ij} denote a correlation coefficient between X_i and X_j . An assignment variable (e_{nm}) is 1 if a VM _{n} is assigned to a PM _{m} , and 0 otherwise. We assume that each VM has a specified QoS to meet and that the QoS can be achieved by allocating enough amounts of resources to the VM. The minimum amounts for achieving the QoS are obtained from (2) with μ_n^k and σ_n^k .

The *minimum resource assignment* (MRA) problem may be formulated as a *multi-capacity stochastic bin packing* (MCSBP) problem:

$$\begin{aligned} & \min \sum_m f_m q_m \\ \text{s.t.} & \begin{cases} q_m = \sum_k w_k r_m^k \\ e_{nm} \in \{0, 1\} \quad \forall m \in \{1, \dots, M\}, \forall n \in \{1, \dots, N\} \\ f_m \in \{0, 1\} \quad \forall m \in \{1, \dots, M\} \\ \sum_{m=1}^M e_{nm} = 1 \quad \forall n \in \{1, \dots, N\} \\ f_m \geq N^{-1} \sum_{n=1}^N e_{nm} \quad \forall m \in \{1, \dots, M\} \\ \text{VAR}_m^k = \sum_{i=1}^N \sum_{j=1}^N e_{im} e_{jm} \rho_{ij} (a_i^k \sigma_i) (a_j^k \sigma_j) \\ \sum_{n=1}^N e_{nm} (a_n^k \mu_n + b_n^k) + \beta \sqrt{\text{VAR}_m^k} \leq r_m^k \\ \forall m \in \{1, \dots, M\}, \forall k \in \{1, \dots, K\} \end{cases} \end{aligned} \quad (5)$$

It is assumed that the following information is given: μ_n , σ_n , w_k , a_n^k , b_n^k , ρ_{ij} , and r_m^k . The objective is to assign VMs to PMs (e_{nm} values – the optimization variable) so as to minimize the total amount of allocated resource while meeting a target QoS value for each VM. If all PMs are homogenous, the objective is simply to minimize the number of active PMs, i.e., $\sum_m f_m$. Note that active means the PM has been assigned at least one VM. This is similar to the objective function used in classical bin packing with identical bin size. However, in order to consider the heterogeneous PMs, we use a new objective function, which is to minimize the sum of total

resource capacities of all active PMs (i.e. $\sum_m f_m \sum_k w_k r_m^k$). Note that a new parameter w_k is introduced to normalize different resource types. The objective implies that resource capacity of a PM will be fully utilized as soon as the PM becomes active even if actual utilization is very small. It implicitly drives a solution to have as few active PMs as possible (i.e., consolidation). There is a couple of important constraints to be met: 1) every VM must be deployed on a PM. 2) the aggregate resource demands of VMs in the same PM do not exceed resource capacity of the PM. This second constraint should hold in order to avoid performance degradation.

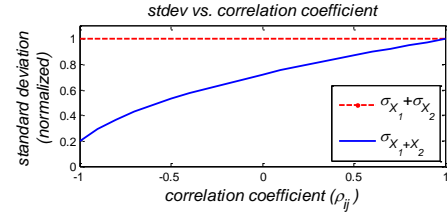


Figure 1 Effect of correlations between RVs on a standard deviation

This MCSBP problem is a variation of the *bin packing* (BP) problem, which is known to be NP-hard [12]. The MCSBP is also NP-hard.

Theorem: The MCSBP problem is NP-hard.

Proof: Consider a special case of the MCSBP problem; standard deviation of the workload intensity is zero, capacity of PMs (size of the bins) is constant, and only one type of resource is to be considered, that is:

$$\sigma_n = 0, r_m^k = \begin{cases} R & \text{if } k=1 \\ 0 & \text{otherwise} \end{cases}, a_n^k = \begin{cases} 1 & \text{if } k=1 \\ 0 & \text{otherwise} \end{cases}, b_n^k = 0 \quad (6)$$

For this special case, the MCSBP problem becomes a classical BP problem:

$$\begin{aligned} & \min \sum_m f_m \\ \text{s.t.} & \begin{cases} e_{nm} \in \{0, 1\} \quad \forall m \in \{1, \dots, M\}, \forall n \in \{1, \dots, N\} \\ f_m \in \{0, 1\} \quad \forall m \in \{1, \dots, M\} \\ \sum_{m=1}^M e_{nm} = 1 \quad \forall n \in \{1, \dots, N\} \\ f_m \geq N^{-1} \sum_{n=1}^N e_{nm} \quad \forall m \in \{1, \dots, M\} \\ \sum_{n=1}^N e_{nm} \mu_n \leq R \quad \forall m \in \{1, \dots, M\}, \forall k \in \{1, \dots, K\} \end{cases} \end{aligned} \quad (7)$$

The BP problem, therefore, is reducible to the MCSBP. Because the BP problem is known to be NP-hard, the MCSBP is NP-hard too. \square

The fact that MCSBP is NP-hard implies that it is very hard to find the optimal solution. A heuristic method, hence, is proposed to solve the problem.

III. HIERARCHICAL RESOURCE MANAGEMENT SOLUTION

In this section the main idea and algorithms for the hierarchical resource management solution is introduced. As depicted in Figure 2, the proposed solution consists of two distinct resource managers: the *global* and *local* managers. The *global* manager assigns VMs to a cluster first, and then the *local* manager deploys the VMs to PMs in the cluster.

Modern data centers consist of many clusters (Figure 2), and the number of PMs in one cluster is bounded because the cluster's resource capacity (e.g., power supply and network bandwidth) is limited. Therefore, a larger data center has more

² More precisely, the mean and variance are supposed to be denoted by μ_{X_n} and $\sigma_{X_n}^2$. However, for the sake of simplicity, the simpler notations are used.

clusters, but typically not bigger clusters. The key advantage of the proposed solution is that it splits a large problem into a number of small problems independent of one another. Because of its small size and independent characteristic, there is an opportunity to apply more sophisticated and elaborate solution approaches, which is not possible for the original (flat) problem because of its very large size.

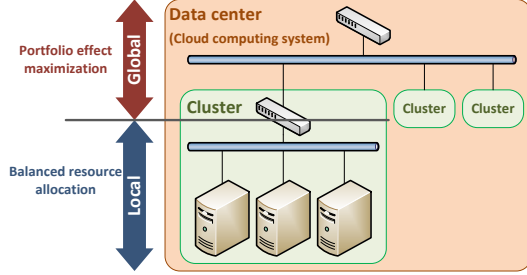


Figure 2 A typical cloud computing system and hierarchical resource management

While a hierarchical approach is scalable, the quality of result from it may be much worse than a flat (non-hierarchical) method. If the original problem is not well divided into sub problems, the quality of the result may be very bad and unacceptable. The *global* manager, therefore, is required to be designed carefully. Likewise, the *local* manager should be designed well to ensure a high quality for the result.

The approach of the global manager is quite different from that of the local manager. The *global* manager maximizes the portfolio effect, i.e., least correlated VMs are deployed on the same cluster. On the other hand, the *local* manager makes the *balanced* usage of resource types within a PM. More detailed explanation of the managers is provided in the following section.

A. Global Resource Manager

The *global* resource manager assigns VMs to clusters. For better quality of the overall result, the proper objective should be chosen. Our objective is to minimize the total resource usage of clusters (8). The resource usage is simply defined as the sum of resource allocation of each type. Note that total mean is always the same regardless of VM deployment methods. On the other hand, the standard deviation is not, that is, it can be decreased using a well-designed method. In this paper, the *cost* is introduced and defined as the sum of the standard deviations ($\sum_{c=1}^C \sum_{k=1}^K \sigma_c^k$). The objective is to minimize this cost. Intuitively, this objective leads to the followings: first, the least correlated VMs are assigned to the same cluster, and the gross amount of resource allocation is minimized. Second, the local manager could assume VMs are uncorrelated, which makes the solution simpler and faster.

$$\min \sum_{c=1}^C \sum_{k=1}^K (\mu_c^k + \beta \sigma_c^k) \equiv \min \sum_{c=1}^C \sum_{k=1}^K \sigma_c^k \quad (8)$$

where $\sigma_c^k = \sqrt{\sum_{i=1}^N \sum_{j=1}^N g_{ic} g_{jc} \rho_{ij} (a_i^k \sigma_i) (a_j^k \sigma_j)}$ and $\mu_c^k = \sum_{n=1}^N g_{nc} (a_n^k \mu_n + b_n^k)$. $g_{nc} = 1$ if VM_n is deployed on c^{th} cluster and 0 otherwise.

Some VMs may be correlated with one another. For example, multiple VMs may be spawned off by the same application or VMs may correspond to different tiers of a multi-tiered application, etc. We can imagine two distinctive situations: all VMs are uncorrelated or some VMs are correlated with one another. Because the uncorrelated case is

simpler, we analyze and solve this case first in order to get some useful intuitions. After that we extend the algorithm for the other situation (correlated).

The key idea in the algorithm is based on the following proposition [14]:

Proposition: Suppose we have N items and C bins. Size of all items is 1 (constant) and cost of each item is σ_n ($n = 1, 2, \dots, N$). The size of the c^{th} bin is r_c (integer) and total size of all bins is the same as the number of items ($\sum_{c=1}^C r_c = N$). The items are sorted in non-increasing order of their costs ($\sigma_i \geq \sigma_j$ for $i < j$). The bins are also sorted in non-increasing order of their sizes ($r_i \geq r_j$ for $i < j$). Set S_c is the set of items put into the c^{th} bin. The overall cost is defined as:

$$\text{overall cost} := \sum_{c=1}^C \sqrt{\sum_{n \in S_c} \sigma_n^2} \quad (9)$$

The cost is minimized if the bigger bin contains the items with greater cost(10).

$$\sigma_a \geq \sigma_b \quad \text{for } a \in S_i \text{ and } b \in S_j \text{ (} i < j \text{)} \quad (10)$$

The proof of the proposition is shown in [14]. Note that the proposition is not perfectly fit our problem because it assumes that the size of items is constant. The proposition, hence, cannot be used without modifications. The modification is made based on a simple intuition as follows; assume that an item is already in a bin and the item will be replaced by other items with smaller size. For the sake of simplicity, the substitutes are assumed to be identical (i.e., their size and cost are all the same). With the further assumption that the ratio of the original item's size to the substitutes' size is integer, we can show that the substitution decreases the overall cost (9) if the following inequality holds:

$$\frac{\sigma_{\text{original}}^2}{\text{size}_{\text{original}}} < \frac{\sigma_{\text{substitute}}^2}{\text{size}_{\text{substitute}}} \quad (11)$$

Hence, the cost of VM_n is redefined (12), and the proposed algorithm assigns VMs to clusters based on this cost.

$$\text{cost}_n := \frac{\sigma_n^2}{\sum_{k=1}^K \mu_{R_n^k} + \beta \sigma_{R_n^k}} = \frac{(\sum_{k=1}^K a_n^k \sigma_n)^2}{\sum_{k=1}^K a_n^k \mu_n + b_n^k + \beta a_n^k \sigma_n} \quad (12)$$

The pseudo code of a VM-to-cluster algorithm (VM2C, preliminary version) is shown below. Its main structure is similar to the *First Fit Decreasing* (FFD) heuristic [15].

VM-to-cluster Algorithm (VM2C): uncorrelated

Inputs: $r_c^k, \mu_n, \sigma_n, a_n^k, b_n^k$, and ρ_{ij}

Output: g_{nc}

- 1: sort clusters C by its size ($\sum_{k=1}^K r_c^k$) in non-increasing order
 - 2: sort VMs by its cost (12) in non-increasing order
 - 3: for each cluster C do
 - 4: for each unassigned VM do
 - 5: $g_{nc} = 1$ // assign VM_n to cluster C
 - 6: $\text{resource}^k = \mu_c^k + \beta \sigma_c^k$ (8)
 - 7: if $\text{resource}^k > r_c^k$ for $\exists k$ then
 - 8: $g_{nc} = 0$ // cancel the assignment
 - 9: end if
 - 10: end for
 - 11: end for
-

Figure 3 VM-to-cluster Algorithm for uncorrelated case

As shown in Figure 3, we sort clusters and VMs in non-increasing order (lines 1 and 2) by cluster's size and VM's cost respectively. The size of a cluster is calculated by

aggregating the total amount of resource types of all PMs in the cluster. For each cluster, the algorithm pre-assigns the VM (lines 4 and 5) with largest cost (12) among all unassigned VMs. It calculates the total amount of resource that the cluster is supposed to provide (line 6). If it is greater than the capacity of the cluster, the assignment is canceled (line 7 and 8). The above steps are repeated until either all VMs are assigned or all clusters are full.

The above algorithm is required to be extended to support the correlated cases. Dealing with the correlated case requires very high amount of computing resources because the complexity of correlation calculation is square of the number of VMs. Finding the optimal solution takes huge amount of time and it may not be practical. Hence, we present a heuristic approach to solve the problem.

The main idea in the heuristic is that a VM is assigned to a cluster one at a time and the best VM is selected in a greedy manner. Suppose that N VMs are already in a cluster and another VM is going to be assigned to the cluster. The new variance of the cluster becomes:

$$(\sigma_{c'}^k)^2 = \sum_{i=1}^N \sum_{j=1}^N g_{ic} g_{jc} \rho_{ij} (a_i^k \sigma_i) (a_j^k \sigma_j) + 2 \left(\sum_{n=1}^N g_{ic} \rho_{nN} a_n^k \sigma_n \right) (a_{N+1}^k \sigma_{N+1}) + (a_{N+1}^k \sigma_{N+1})^2 \quad (13)$$

The increase in variance ($\Delta_{c,N+1}^k$) is shown in (14). The *overhead* is defined as the ratio of the amount of increase to the total resource demands of VM $_{N+1}$ (15).

$$\Delta_{c,N+1}^k = 2 \left(\sum_{n=1}^N g_{ic} \rho_{nN} a_n^k \sigma_n \right) (a_{N+1}^k \sigma_{N+1}) + (a_{N+1}^k \sigma_{N+1})^2 \quad (14)$$

$$\text{overhead}_{c,N+1} := \frac{\sum_{k=1}^K \Delta_{c,N+1}^k}{\sum_{k=1}^K (a_{N+1}^k \mu_{N+1} + b_{N+1}^k + \beta a_{N+1}^k \sigma_{N+1})} \quad (15)$$

The final version of the VM2C algorithm (Figure 4) is nearly identical to the preliminary version (Figure 3) except for a few lines (lines 5 through 7). The algorithm considers the first \mathbf{W} (it is called ‘window size’) VMs as candidates for allocation, and choose one of which *overhead* (15) is smallest among the candidates.

VM-to-cluster Algorithm (VM2C): correlated

Inputs: $r_c^k, \mu_n, \sigma_n, a_n^k, b_n^k$, and ρ_{ij}

Output: g_{nc}

```

1: sort clusters  $C$  by its size ( $\sum_{k=1}^K r_c^k$ ) in non-increasing order
2: sort VMs by its cost (12) in non-increasing order
3: for each cluster  $C$  do
4:   while true then
5:     for the first  $\mathbf{W}$  unassigned VMs do
6:       find a VM of which overhead (15) is smallest
7:     end for
8:      $g_{nc} = 1$  // assign VM $_n$  to cluster  $C$ 
9:      $\text{resource}^k = \mu_c^k + \beta \sigma_c^k$  (8)
10:    if  $\text{resource}^k > r_c^k$  for  $\exists k$  then
11:       $g_{nc} = 0$  // cancel the assignment
12:      break // break while loop
13:    end if
14:  end while
15: end for
```

Figure 4 VM-to-cluster Algorithm (final version)

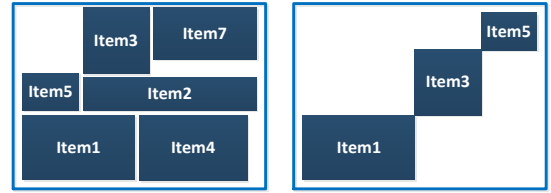
It is important to choose a proper window size (\mathbf{W}) for better results; Bigger \mathbf{W} may produces higher quality of result, but at the same time, increases the execution time of the

algorithm. It is not simple to find the best \mathbf{W} ; this topic will be discussed at Section IV.

B. Local Resource Manager

The *local resource manager* deploys VMs, which are assigned to a cluster by the *global manager*, on PMs in the cluster. The problem of local level is identical to the original one (5) except for the problem size; its size is much smaller than that of the original problem. The local-level problem is also NP-hard, so we propose a heuristic algorithm.

As mentioned before, the problem can be formulated into the MCSBP problem. Note that a *multi-capacity bin packing* (MCBP) problem is different from a classical *multi-dimensional bin packing* (MDBP) problem; the difference is depicted in Figure 5. In this example, each bin and item has a pair of capacity (horizontal and vertical) information. In a MDBP problem, an item can be put into a bin if there is enough geometric space for the item. The MCBP problem, however, differs from the MDBP problem; once an item deployed on a bin, the corresponding capacity is dedicated to the item and not available to other items.



(a) multi-dimensional bin packing

(b) multi-capacity bin packing

Figure 5 Multi-dimensional vs. multi-capacity bin packing

Nilabja *et al.* propose several heuristics for solving the MCBP[16]. The heuristics are very efficient but the quality of results is less than satisfactory. William *et al.* propose *permutation pack* (PP) and *choose pack* (CP) heuristics [17]. The quality of results from the PP is quite good but its complexity increases exponentially as the number of resource types grows. CP is simpler than the PP but the quality of output is worse. In this paper we present a heuristic algorithm, which is better than PP in terms of both the quality of results and algorithm’s running time.

The key idea in the proposed algorithm is to achieve *balanced* resource allocation: deploy a VM to a PM where amount of available resources is most ‘similar’ to that of the VM. A detailed explanation of ‘similarity metric’ is presented below. First, two resource vectors are defined (16): \vec{r}_n and \vec{a}_m . \vec{r}_n is a resource demand vector of VM $_n$ and \vec{a}_m is an available resource vector of PM $_m$.

$$\vec{r}_n := (r_1, r_2, \dots, r_K), \vec{a}_m := (a_1, a_2, \dots, a_K) \quad (16)$$

where $r_k = a_n^k \mu_n + b_n^k + \beta a_n^k \sigma_n$, and a_k is the amount of available resource of a PM $_m$. The *difference* between the PM and VM is defined as the distance of two unit-sized vectors (17). The algorithm selects the PM with the minimum *difference*, which means the PM is most ‘similar’ to the VM. The basic structure of the VM-to-PM algorithm (VM2PM) is similar to the best fit decreasing (BFD) algorithm. For us, *best* means the smallest *difference* (17).

$$\text{difference} := d_{nm} = \left\| \frac{\vec{r}_n}{\|\vec{r}_n\|} - \frac{\vec{a}_m}{\|\vec{a}_m\|} \right\| \quad (17)$$

There may be no feasible solution at the local level. If the *local manager* cannot deploy some VMs, it gives the list of those VMs to the *global manager*. The *global manager* assigns

these VMs to other clusters. These steps are repeated until either all VMs are assigned or all clusters are full.

C. Flat VM Allocation Algorithm

A hierarchical solution is generally scalable, but the quality of result may be worse than that of a flat solution. For the purpose of comparison, we develop a flat version of VM allocation algorithm, which conceptually merges VM2C (*global*) and VM2PM (*local*) algorithms. First, find W_1 candidate VMs of which *difference* is smallest (line 4). Among the candidates, choose the W_2 VMs with the least cost (12) (line 5 and 6). Finally, pick the VM with the smallest overhead (15) among the W_2 VMs, and deploy the VM to the PM.

Flat VM Allocation Algorithm (FLAT)

Inputs: $r_m^k, \mu_n, \sigma_n, a_n^k, b_n^k$, and ρ_{ij}

Output: e_{nm}

```

1: sort PM by its size ( $\sum_{k=1}^K r_m^k$ ) in non-increasing order
2: for each PM do
3:   do while the PM is not fully assigned
4:     find  $W_1$  candidate VMs with smallest difference (17)
5:     sort the VMs by its cost (12) in non-increasing order
6:     for the first  $W_2$  candidate VMs do
7:       find a VM of which overhead (15) is smallest
8:     end for
9:      $e_{nm} = 1$  // assign VMn to PMm
10:    if PMm is full (cannot meet QoS)
11:       $e_{nm} = 1$  // cancel the assignment
12:      break // break while loop
13:    end if
14:  end for

```

Figure 6 Flat VM allocation algorithm

IV. SIMULATION RESULTS

A. Simulation setup

For the simulation we needed the following data: 1) capacity of PMs, 2) list of PMs in clusters, 3) resource demands of VMs, and 4) the correlation coefficients among VMs. The data is randomly generated based on the following information:

- Number of VMs (N), PMs (M), and clusters (C)
- Workload intensity of VMs: upper and lower bounds of μ_n and σ_n
- Scale factors of resource: upper and lower bounds of a_n^k and b_n^k (the same bounds are applied to all VMs)
- Capacity of PMs: upper and lower bounds of r_m^k (the same bounds for all PMs)
- Dimension of resource (K): the number of resource types

It is also randomly decided which PMs are placed in which clusters (topology). Because VMs are heterogeneous, workload intensity is randomly generated using the given bounds. Likewise, the scale factors and PM capacity are randomly generated.

Making a valid correlation coefficient matrix $[\rho_{ij}]$ is important, so we use the *hypersphere decomposition* [18] method, which is a relatively simple method for generating a valid correlation matrix.

B. Global Resource Manager

The objective of the *global* resource manager is to minimize the sum of standard deviations of clusters (8), which will be called *cost* in this section; lower cost means better quality of the solution.

To assess the quality of solutions generated by the proposed algorithms, our solution is compared with some other well-known methods:

- **SA** – use a *simulated annealing algorithm* [19]. It does not guarantee to find the global optimal point, but it finds a near-optimal. This method may generate different solutions each time, so we run the SA six times and pick the ‘best’ result. SA takes very long time to find a solution and it is not practical, but it can be a good indicator of determining the quality of other algorithms.
- **random** – assign randomly. If a solution is worse than *random*, it implies the quality of the solution is quite poor. We run *random* algorithm ten times and report the ‘average’ of these runs.
- **FFD** – use *First Fit Decreasing* algorithm, which is a well-known and commonly used heuristic to solve the bin packing problem. It treats the problem as a single capacity bin packing problem by assuming there is only one type of resource; it uses an aggregate amount of all kinds of resource.
- **PM2C** – use the PM-to-cluster algorithm (c.f. Figure 4)

We first investigate the quality of the algorithms by comparing them with the SA that produces the near optimal solution. Normalized cost of the algorithms is shown in Figure 7. For a fair comparison, we generate eight different test cases based on the same setup and run simulations. Note that even with the same setup, the results can be different because the data is randomly generated in each case (c.f. Section IV-A).

As depicted in Figure 7–a, PM2C produces the best result, which is even better than SA. The cost of *random* is biggest as expected. FFD is better than *random* but its cost is around 30% greater than PM2C. The same trend is observed for $K=7$ (Figure 7–b).

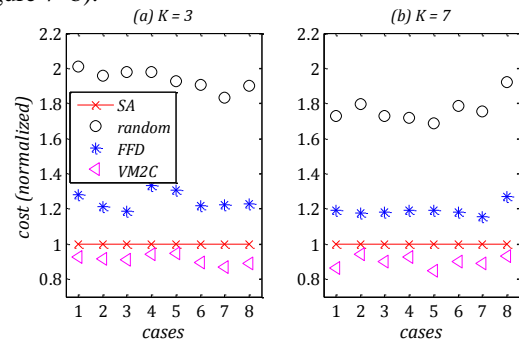


Figure 7 Quality comparison among the algorithms (N=500, W=50)

The cost of result and running time from four different N values are presented in Figure 8. The original data has 2000 VMs in it and the other three data are made from the original one by random sampling. The same trend is observed in both cost and running time for four N values: 1) the rate of reduction in the cost decreases as W increase. 2) running time is linearly increasing as W increase. Note that the slope of smallest N (=500) is most steep, which is because growth in running time is relatively bigger compared to the running time itself.

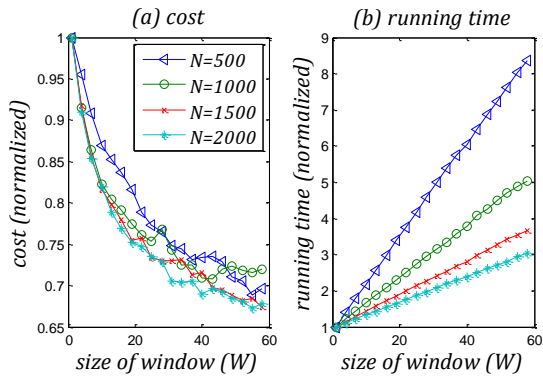


Figure 8 Cost and algorithm running time vs. window size (W)

As mentioned before, choosing the proper window size (W) of PM2C is quite important. Unnecessarily large W is not acceptable due to very long running time. It is hard to find the optimal window size from an analytical way. Instead, we propose a simple way to find the proper window size; as shown in Figure 8, relationship between cost and window size is very similar among the four cases. First, make a smaller problem by randomly select VMs and PMs from the original problem (random sampling). Next, run PM2C multiple times with different W and find the best W . As shown in Figure 9, running time is deeply dependent on the problem size (N), so investigation of smaller problem can find a proper window size in a shorter time.

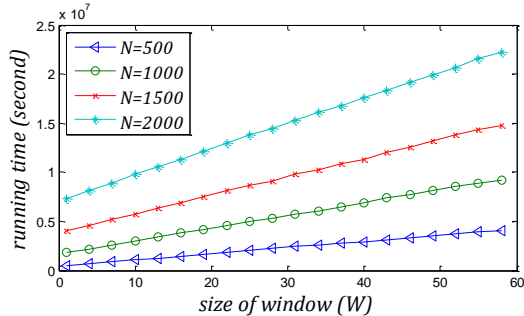


Figure 9 Running time vs. W for different problem size (N)

C. Local Resource Manager

The local resource manager deploys VMs to PMs. In order to assess the quality of results from the proposed algorithm, it is compared with the below heuristics:

- **Permutation Pack (PP)** [17] – for each bin, find an item whose ordering of the resource demands best matches the ordering of residual resources in the bin. This algorithm only considers ‘ w ’ resource types which are mostly important, and the window size (w) is the same as dimension of resource (K) unless it is specified. The complexity of PP is $O\left(\frac{MNd!}{(d-w)!}\right)$, and this increases rapidly as resource dimension (K) increases.
- **FFD** – use *First Fit Decreasing* algorithm. It is the same as the FFD presented in Section IV-B.
- **VM2PM** – use the VM-to-PM algorithm (VM2PM).

The VM2PM generates the best quality of result (c.f. Figure 10). PP is the second best and both $PP_{w=3}$ and FFD are the worst. FFD is a simple algorithm and its running time is smallest. Running time of PP exponentially increases as resource dimension increases. Figure 10 implies that PP is not applicable when K is greater than 8 because of huge running time. On the other hand, the proposed algorithm, VM2PM,

could be used even if K is pretty large. Note that $PP_{w=3}$ is supposed to be simpler than PP because it considers the order of three biggest resources only, but the running time of $PP_{w=3}$ is larger than that of PP when K is less than four. It is caused by a way of implementation; the algorithm makes a list for comparison of the order, and revises the list only when window size (w) is specified. Because of this additional computation, the running time of $PP_{w=3}$ is greater than PP for $K < 4$.

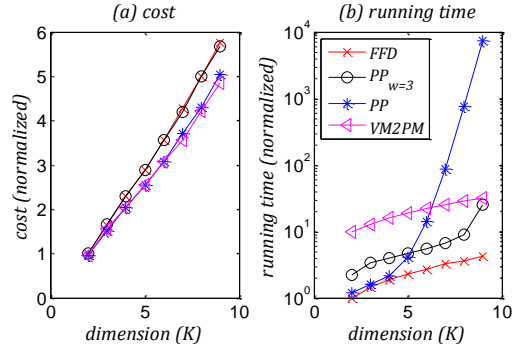


Figure 10 Cost and running time vs. dimension of resource (K)

D. Overall performance comparisons

We have shown that the proposed algorithms are simple and work well for both global and local resource allocations. However, the high quality of each level does not necessarily guarantee the high quality of the overall solution. In this section, we will compare the quality of the final solution generated by different algorithms and verify that the proposed algorithms produce high quality solutions. In addition to this, we run the simulation for a large number of VMs, and show if the proposed scheme is scalable or not. In this section the results of below heuristics are presented:

- **FINAL** – use the proposed hierarchical algorithms: VM2C and VM2PM algorithms.
- **PP** – use permutation pack algorithm. It is non-hierarchical method.
- **FFD** – use *First Fit Decreasing* algorithm. It is the same as the FFD presented in Section IV-B. It is non-hierarchical.
- **FLAT** – use the flat VM assignment algorithm (Figure 6). W_2 is equal to W of FINAL and $W_1 = 5W_2$

For the simulation K is set to 7; if K is greater than 7 running time of PP become very large (Figure 10) even for small N . Comparison among the algorithms with different numbers of VMs (N) is reported in Figure 11. It is seen that costs of FFD and $PP_{w=3}$ are much larger than those of all the others. For the all cases FLAT produces the best results. PP and FLAT produce similar quality of results. The difference in cost between FLAT and FINAL is less than 4%; which implies the quality of FINAL’s result is quite similar to that of FLAT.

The scalability is the one of the most important features of the resource management solution for the cloud computing. Hence, the relationship between running time of the algorithms and problem size is very important. The running time of the proposed algorithm is calculated as follow:

$$T_{\text{running time}} = T_{\text{global}} + \max(T_{\text{local}}) \quad (18)$$

where T_{global} and T_{local} are running time of the *global* resource manager and *local* resource managers, respectively. The *local* managers run in parallel; therefore, the longest execution time among them is considered for the calculation.

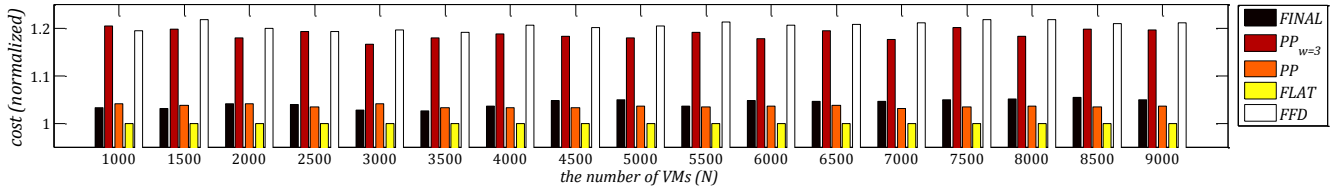


Figure 11 Quality comparison among the algorithms

The running time comparison among the algorithms is shown in Figure 12. Both PP and FLAT run for a very long time to find a solution for the large number of VMs because they are non-hierarchical methods which are not scalable in general. Because of their large running time, both PP and FLAT may not be applicable to the large-sized problems. On the other hand, the running time of all other heuristics is acceptable. Note that the increase rate in running time of FINAL is smallest among the algorithms; this is one of the key benefits from a hierarchical method. The quality of the FFD's result is too low as shown in Figure 11 (20% greater than FLAT), so FFD is not acceptable either. For the same reason, $PP_{w=3}$ is not a good solution. Hence, FINAL is the best method among these algorithms and its cost is around 15% less than FFD and $PP_{w=3}$.

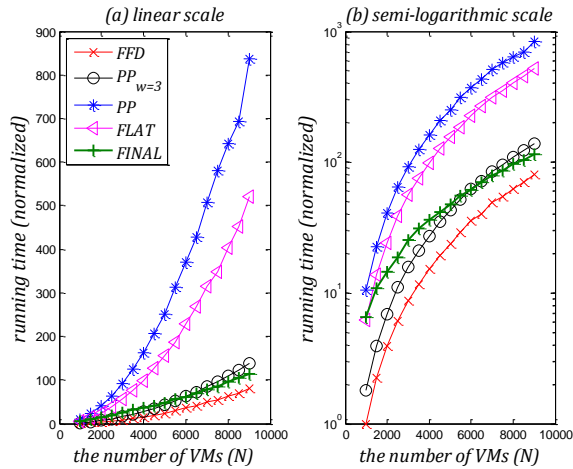


Figure 12 Running time comparison among the algorithms

V. CONCLUSIONS

With increasing energy cost of cloud computing systems, necessity of energy aware resource management techniques has been growing. This paper proposed a hierarchical resource management solution which produces high quality solutions and is scalable for a number of resource types as well as a number of VMs. The resource demands are modeled as random variables and correlation among these RVs are considered. The proposed solution outperforms a well-known heuristic algorithm; it achieves around 15% cost reduction compared to FFD with acceptable running time.

REFERENCES

- [1] Fehrenbacher, K. (2012) The era of the 100 MW data center.
- [2] Cook, G. and Horn, J. V. (2011) How dirty is your data. *A Look at the Energy Choices That Power Cloud Computing*.
- [3] Barroso, L. A. and Holze, U. (2007) The case for energy-proportional computing. *Computer*. **40**(12) 33-37.
- [4] Mehta, S. and Neogi, A. (2008) Recon: A tool to recommend dynamic server consolidation in multi-cluster data centers. Network Operations and Management Symposium, 2008. NOMS 2008. IEEE. IEEE.
- [5] Christopher, C., Keir, F., Steven, H., Jacob Gorm, H., Eric, J., Christian, L., Ian, P. and Andrew, W. (2005) Live migration of virtual machines. Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2 USENIX Association.
- [6] Jon, K., Yuval, R. and va, T. (2000) Allocating Bandwidth for Bursty Connections. *SIAM J. Comput.* **30**(1) 191-217.
- [7] Meng, W., Xiaoqiao, M. and Li, Z. (2011) Consolidating virtual machines with dynamic bandwidth demand in data centers. INFOCOM, 2011 Proceedings IEEE.
- [8] Ming, C., Hui, Z., Ya-Yunn, S., Xiaorui, W., Guofei, J. and Yoshihira, K. (2011) Effective VM sizing in virtualized data centers. Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on.
- [9] Akshat, V., Gargi, D., Tapan Kumar, N., Pradipta, D. and Ravi, K. (2009) Server workload analysis for power minimization using consolidation. Proceedings of the 2009 conference on USENIX Annual technical conference USENIX Association.
- [10] Xiaoqiao, M., Canturk, I., Jeffrey, K., Li, Z., Eric, B. and Dimitrios, P. (2010) Efficient resource provisioning in compute clouds via VM multiplexing. Proceedings of the 7th international conference on Autonomic computing ACM.
- [11] Breitgand, D. and Epstein, A. (2012) Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. INFOCOM, 2012 Proceedings IEEE.
- [12] Billingsley, P. (2012) Probability and Measure Wiley.
- [13] Rudd, A. and Clasing, H. K. (1988) Modern portfolio theory: the principles of investment management Andrew Rudd.
- [14] Hwang, I. and Pedram, M. (2012) Portfolio Theory-Based Resource Assignment in a Cloud Computing System. Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on.
- [15] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001) Introduction To Algorithms MIT Press.
- [16] Nilabja, R., John, S. K., Nishanth, S., Gautam, B. and Douglas, C. S. (2008) Toward Effective Multi-Capacity Resource Allocation in Distributed Real-Time and Embedded Systems. Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing IEEE Computer Society.
- [17] William, L., George, K. and Vipin, K. (1999) Multi-Capacity Bin Packing Algorithms with Applications to Job Scheduling under Multiple Constraints. Proceedings of the 1999 International Conference on Parallel Processing IEEE Computer Society.
- [18] Rebonato, R. and Jäckel, P. (2000) The most general methodology for creating a valid correlation matrix for risk management and option pricing purposes. *Journal of Risk*.
- [19] Christopher, C. S., cim and Bruce, L. G. (1983) Optimization by simulated annealing: A preliminary computational study for the TSP. Proceedings of the 15th conference on Winter Simulation - Volume 2 IEEE Press.