Monte Carlo Based Server Consolidation for Energy Efficient Cloud Data Centers

Bryan Harris and Nihat Altiparmak
Department of Computer Engineering & Computer Science
University of Louisville, USA
{bryan.harris.1,nihat.altiparmak}@louisville.edu

***Abstract*—The growing energy consumption of data centers is a compelling global problem and effective server consolidation is at the heart of energy efficient cloud data centers. A variant of bin packing can be used to model the server consolidation problem, where the constraints are multidimensional and heterogeneous vectors rather than scalars and the goal is to satisfy the requested resource allocation using the minimum number physical servers. Since bin packing is NP-hard, we rely on heuristics for practical solutions. Variations of First Fit Decreasing (FFD) based heuristics have been shown to be effective both in theory and practice for the one dimensional homogeneous case. However, the multidimensional and heterogeneous aspects of the server consolidation problem make it more complicated, requiring additional research to adapt FFD to the server consolidation problem. In this paper, we present a new FFD-based server consolidation technique using a Monte Carlo method and Shannon entropy, which considers resource bottlenecks and dynamically adjusts to variance in the utilization of different resources. The proposed heuristic outperforms existing techniques in all scenarios, achieving within 2-5% of optimal on average for medium to high variance in resource utilization, and within 10% worse than optimal on average for all scenarios.**

***Index Terms*—bin packing; server consolidation; data centers**

I. Introduction

Users and enterprises heavily utilize cloud-based services instead of purchasing and managing their own hardware. As a result, the energy consumption of data centers has become an important issue. In 2015, the worldwide annual energy consumption of data centers was estimated to be around 416 TWh (roughly $55 billion) [1]. This is much as the annual energy consumption of large industrialized countries like UK and France, according to the CIA World Factbook [2]. Furthermore, the energy cost of data centers is expected to double every five years [3]. In addition to its economic burden, this consumption has a negative impact on the environment. According to the Environmental Protection Agency (EPA), generating 1 KWh of electricity results in an average of 1.55 pounds of carbon dioxide emissions. With more than 400 TWh of yearly consumption, improvements in the energy efficiency of data centers can have a massive economic and environmental impact on society. The following quote from Eric Schmidt, the former CEO of Google, emphasizes the importance of energy-efficient data centers from the perspective of cloud providers: *"What matters most to the computer designers at Google is not speed but power—low power, because data centers can consume as much electricity as a city."* Amazon estimates the energy-related costs of its data centers as 42% of their total budget, including both direct power consumption and cooling cost [3].

Virtualization is a proven resource sharing technology used in cloud architectures, and effective server consolidation is at the heart of energy efficient cloud data centers. By packing virtual machines into the fewest possible physical machines, an effective server consolidation allows unused physical machines to be switched to low power states and can achieve significant energy savings [4]. The US Department of Energy estimated that with effective server consolidation, the energy consumption of data centers can be reduced by around 520 TWh within the five-year period of 2015 and 2020 [5].

Server consolidation in cloud data centers can be formulated as a variant of the bin packing problem, where items and bins are multidimensional and heterogeneous vectors representing resource demands and available server capacities. In bin packing theory, first fit decreasing based heuristics are proven to be effective with near optimal solutions in the one dimensional homogeneous bin case [6]; however, sorting multidimensional items and bins is a difficult task where dimensions have incomparable resources such as CPU cores, memory, network bandwidth, etc. An additional complexity is that bins have different (heterogeneous) capacities. Therefore an efficient dimensionality reduction technique that converts multidimensional vectors to sortable scalars, as well as handling the heterogeneity of bins, is necessary in order to achieve a tight packing. Existing approaches either ignore the heterogeneity of bins, or perform dimensionality reduction based on predetermined resource utilization assumptions prone to generate sub-optimal solutions. Therefore, new research is necessary to perform better server consolidation and achieve improved energy efficiency in cloud data centers.

In this paper, we propose a new server consolidation heuristic that can perform near optimal resource allocation in cloud data centers for reduced energy consumption. The proposed approach can adjust dynamically to resource utilization variance of different dimensions and automatically predict resource bottlenecks based on a Monte Carlo method and Shannon entropy. We theoretically formulate and solve the optimal server consolidation problem using a linear programming model and provide extensive performance evaluation by analyzing low, medium, and high resource utilization variances, as well provide comparisons with optimal values.

2330-2186/19/$31.00 ©2019 IEEE
DOI 10.1109/CloudCom.2019.00046

II. Background, Motivation, and Related Work

In this section, we first provide the preliminaries of bin packing and server consolidation. Next, we present the motivation and related work.

A. Bin packing

Many resource allocation problems can be modeled as a variant of bin packing. In order to clearly define the problem with its realistic constraints and show its difficulty, this section summarizes the variants of bin packing that are related to the server consolidation problem.

1) Classical Bin Packing (BP): Given a list I of m real numbers $I_1, I_2, I_3, \ldots, I_m$, where each $I_i \in (0, 1]$ represents the size (demand or requirement) of an item, the classical Bin Packing (BP) problem aims to pack all items I into the minimum number of unit capacity bins so that the sum of the sizes of all items packed into any given bin does not exceed one [7]. An inexhaustible supply of bins are assumed to be available for packing and all bins are assumed to be initially empty. Classical BP is one of the most well known NP-hard problems in the field of combinatorial optimization [8].

2) Bin Packing with Heterogeneous Bins (BPHB): Bin Packing with Heterogeneous Bins (BPHB) is one generalization of the classical BP problem with a finite collection of bins allowed to have different capacities. This is in contrast to classical BP where the supply of bins is inexhaustible and all bins start with unit capacity. This generalization is also referred to as the Variable Sized Bin Packing problem, and is NP-hard as it reduces to the classical BP when only one bin size is permitted [9].

3) Vector Bin Packing (VBP): Vector Bin Packing (VBP) [10] is another generalization of the classical BP problem in which both item sizes and bin capacities are d-dimensional vectors rather than scalars. The bins are homogeneous and an inexhaustible supply of them are available for packing. A valid packing in the d-dimensional case requires that the vector sum of the sizes of all items packed into any given bin does not exceed the unit vector. VBP is NP-hard for every d [11], [12], [13].

4) Vector Bin Packing with Heterogeneous Bins (VBPHB): Vector Bin Packing with Heterogeneous Bins (VBPHB) combines BPHB and VBP in a single problem such that bins are allowed to have different vector capacities [14]. For VBPHB, in addition to a list of m items $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \ldots, \mathbf{I}_m$ representing item sizes as vectors, we are also given a list of n bins $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3, \ldots, \mathbf{C}_n$ representing bin capacities as vectors. Unlike BP, BPHB, and VBP, VBPHB does not assume an inexhaustible supply of bins; instead, a solution must place all items into a subset of the provided n bins in order to be valid.

B. Server Consolidation in the Cloud

In virtualized data centers, such as those using a cloud architecture, server consolidation involves the Virtual Machine Placement (VMP) problem, where we wish to assign Virtual Machines (VMs) with multidimensional resource requirements, such as CPU and memory, to the fewest possible Physical Machines (PMs) or servers with sufficient available resources. For energy efficiency, PMs that are not assigned VMs can be switched to a low power state. Various power states are defined by the ACPI standard and switching power states is managed directly by the operating system [15]. Consolidating VMs into the fewest PMs is a common technique and can provide a significant reduction in data center energy consumption [4]. For the rest of this paper, we simply model server consolidation in the cloud as the VMP problem.

Considering energy efficiency as the main objective, VMP is defined as follows: Given m virtual machines $\mathit{VM}_1, \mathit{VM}_2, \ldots, \mathit{VM}_m$ with resource requirements (CPU cores, memory, network bandwidth, etc.) and n physical machines $\mathit{PM}_1, \mathit{PM}_2, \ldots, \mathit{PM}_n$ with resource capacities for d resource types, VMP aims to map VMs to PMs by satisfying the resource demands of the VMs, respecting the resource constraints of the PMs, and minimizing the number of PMs used in the mapping. A single PM may host multiple VMs as long as it respects the resource constraints. PM capacities are expected to be heterogeneous as they represent the available resources of PMs. VMP is an NP-hard problem and equivalent to VBPHB, where VMs represent items, PMs represent bins, and resource types represent dimensions [14]. VMP can be expressed in a linear form as follows:

$$\textbf{Minimize:} \quad \sum_{j=1}^{n} X_j$$

$$\textbf{Subject to:} \quad \sum_{j=1}^{n} B_{ij} = 1; \qquad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} (D_{ik} B_{ij}) \leq C_{jk}; \qquad \begin{cases} j = 1, \ldots, n \\ k = 1, \ldots, d \end{cases}$$

Each B_{ij} is a binary variable that is set to 1 if VM_i is mapped to PM_j, or otherwise set to 0. Therefore B_{ij} represents the final mapping. The first constraint ($\sum_{j=1}^{n} B_{ij} = 1$) ensures that every VM is only mapped to a single PM. C_{jk} is the available capacity of PM_j for resource type k. The sum $\sum_{i=1}^{m} (D_{ik} B_{ij})$ represents the resource usage of PM_j, since D_{ik} (a predefined constant) is the resource demand of VM_i. Finally, the objective function minimizes the number of PMs used in the placement. This is guaranteed using a binary indicator variable X_j, which is set to 0 if $\sum_{i=1}^{m} B_{ij} = 0$, and set to 1 if $\sum_{i=1}^{m} B_{ij} > 0$. Therefore X_j indicates whether PM_j is used in the placement or not, and $\sum_{j=1}^{n} X_j$ is the number of PMs used in the placement. The mapping represented by the B_{ij} values is guaranteed to use the minimum number of PMs satisfying the given resource constraints.

This formulation uses mn B_{ij} variables and n X_j variables. All variables are binary and the total number of unique variables is $n(m + 1)$. In addition, it uses a total of $m + n$ constraints. This is an Integer Linear Programming (ILP) formulation, which is classified as NP-hard [16] and can be solved optimally using an LP solver; however, this approach

is expected to be prohibitively slow beyond m and n of a few hundred [12].

The classical BP problem and its generalizations that we have covered in this paper (including VMP) are summarized in Table I with their item and bin characteristics.

Problem	Item size	Bin capacity
BP	$I_i \in (0,1]$, $i \in \{1,\ldots,m\}$	$C = 1$, infinite bins
BPHB	$I_i \in (0,1]$, $i \in \{1,\ldots,m\}$	$C \in (0,1]$, infinite bins
VBP	$\mathbf{I}_i \in (0,1]^d$, $i \in \{1,\ldots,m\}$	$\mathbf{C} = 1^d$, infinite bins
VBPHB	$\mathbf{I}_i \in (0,1]^d$, $i \in \{1,\ldots,m\}$	$\mathbf{C}_j \in (0,1]^d$, $j \in \{1,\ldots,n\}$
VMP	$\mathbf{I}_i \in \mathbb{R}^d_{>0}$, $i \in \{1,\ldots,m\}$	$\mathbf{C}_j \in \mathbb{R}^d_{>0}$, $j \in \{1,\ldots,n\}$

TABLE I: Bin packing and its generalizations

A problem instance of VBPHB or VMP is *infeasible* if a placement for *all* VMs or items does not exist. This is possible, for example, in cases with too few bins, especially high contention for a particular resource, or simply where total demand exceeds total availability. In such cases, obviously no algorithm or heuristic will find a solution. However, it is also possible for certain heuristics to be unable to find a solution despite the problem instance being feasible.

In addition, VBP heuristics may require preprocessing such as normalization of item sizes and bin capacities in order to provide a better solution, which can be calculated independently for each dimension, by mapping the bin capacity in that dimension to 1 and the item sizes in that dimension to a size no greater than one, in proportion with the bin capacity.

C. Motivation

In bin packing theory, First Fit Decreasing (FFD)-based heuristics are known to be effective both in theory and in practice for the classical one-dimensional homogeneous BP case. FFD guarantees to find an allocation with at most $\frac{11}{9}OPT+1$ bins [7], [6]. The basic idea in FFD is to sort items by their size and pack them from largest to smallest into the first bin in which it will fit. However, the multidimensional and heterogeneous aspects of VMP make it more complicated than one-dimensional homogeneous BP, requiring adaptations in the FFD technique. For heterogeneous bins, a natural approach is to sort them in decreasing order as well, based on some criteria such as their remaining capacity or their likelihood to be used (popularity) in packing. Handling the multidimensionality of VBP and VMP is more complicated since both items and bins are vectors, composed of multiple incomparable dimensions such as CPU cores, memory sizes, network bandwidths, etc. An effective dimensionality reduction technique is necessary to convert vectors into sortable scalar values.

D. Related work

VBP was first investigated for homogeneous bins by Kou et al. [11]. In this work, the authors proposed three simple dimensionality reduction techniques for FFD: Lexicographical (*Lex*), Maximum Component (*MaxComp*), and Maximum Sum (*MaxSum*), which sorts items based on the lexicographical comparison of each dimension, maximum size in any dimension, and the sum of all dimensions, respectively. Maruyama et al. later proposed multiple follow up techniques such as *ExpSum* that sorts items according to the sum of exponents of dimensions, *AvgSum* that sorts items according to a weighted sum of dimensions, and *Prod* that sorts items according to the product of dimensions [17].

With the advent of cloud computing and virtualized data centers in general, VBP received more recent attention. Various recent works adapted these dimensionality reduction techniques for the VMP problem [13], [14], [18], [19], [20]. Panigrahy et al. proposed *DotProd*, a greedy bin-centric heuristic for which item ordering is based on a weighted dot product of the item demand and remaining bin capacity vectors [13]. Jangiti et al. proposed an aggregated ranking technique called *AR*, which first ranks items for each dimension separately, and then aggregates these rankings by summing them to determine an item ordering [20]. However, as both these techniques were intended for VBP with homogeneous bins, when applied to VBPHB, they are unaware of the diversity of remaining capacities of heterogeneous bins. Gabay et al. generalized *DotProd* for VBPHB and proposed three variants with different weight vectors: i.) none, or weight of 1; ii.) weight based on both bin capacity and item requirements; and iii.) weight based solely on bin capacity [14]. The goal of Panigrahy's *DotProduct* for VBP was to minimize the number of bins used; however, the goal of Gabay's problem was to maximize the number of problem instances in which a feasible packing for all items is found. Gabay's generalized *DotProd* technique, *DP (Gabay)*, has not yet been evaluated for the server consolidation problem (VMP), where the aim is to minimize the number of bins used in packing.

In the VMP problem, resource utilization of the dimensions considered for the entire data center, such as CPU, memory, and network, changes dynamically and it is generally not possible to predict bottlenecks in advance. Previously proposed techniques do not consider the intricate interaction between resource utilization of multiple dimensions, and therefore fail to perform effective server consolidation when the resource utilization of dimensions varies over time or deviates from assumptions. For example, *Lex* assumes that the first dimension is most likely to be the bottleneck, while *MaxComp* prioritizes resource allocation based on the dimension with the maximum resource demand for each item. *Prod* assumes that every resource will be utilized evenly, while the *AvgSum* variants try to estimate resource bottlenecks in advance using predetermined weights. On the other hand, our proposed *MonteCarlo* technique adjusts dynamically to resource utilization variations because it requires no prior assumptions of resource importance; the intricate effects of interactions between resources are discovered through random sampling.

III. Monte Carlo Based Server Consolidation

The primary issue when generalizing greedy heuristics such as first fit decreasing (FFD) from one to multiple resources is how to sort items and bins in "decreasing" order. For one dimensional BP, item demands are scalars, so determining the "largest" item or bin is trivial; however, with vector demands

and capacities, the notion of "largest" is less straightforward. Techniques such as *Prod*, *AvgSum*, *MaxComp*, *Lex*, etc., are based on mathematical transformations of these vectors to sortable scalars. In our Monte Carlo based technique, we try to reduce this dimensionality to item and bin rankings by looking at more fundamental properties of restrictiveness and potential, based on measurements of random sampling.

The fundamental property of a "large" *item* is that its resource demands are restrictive—it has fewer options as to where it can be placed than a "smaller" item. Where an item can be placed is not only a function of its demands, but also of its interactions with other items. For example, suppose that an item fits within the capacities of two bins, but another more restrictive item may only fit in one of these bins; this reduces the options for the first item. In our proposed *MonteCarlo* heuristic, we wish to rank items based on the restrictiveness of their placement options while also considering such interactions. We estimate this restrictiveness using information gained from random sampling, as measured by Shannon entropy.

A "large" *bin* is one with high resource capacities, but more specifically one that has the option or potential for holding greater quantities of items or more restrictive items. Using the same random sampling data acquired to sort items, we sort bins by the *popularity* of samples.

Algorithm 1 *MonteCarlo* based server consolidation

In: bin capacities $\mathbf{C}_b$, item demands $\mathbf{I}_i$,
number of bins n, number of items m, number of samples k
Out: assignment $\mathbf{P}: i \to b$

initialize $m \times n$ matrix $\mathbf{M}$ to zero
for *sample* **in** $\{1, \ldots, k\}$ **do**
  $\mathbf{B}_b \leftarrow \mathbf{C}_b \quad \forall b \in \{1, \ldots, n\}$ // copy bins
  for all i **in** randomized $(1, \ldots, m)$ **do**
    for all b **in** randomized $(1, \ldots, n)$ **do**
      if $\mathbf{I}_i \leq \mathbf{B}_b$ for all d dimensions **then**
        $\mathbf{M}_{i,b}$ += 1
        $\mathbf{B}_b$ −= $\mathbf{I}_i$
        break // next item i
for i **in** $\{1, \ldots, m\}$ **do**
  rowsum $\leftarrow \sum_{j=0}^{n} \mathbf{M}_{i,j}$
  $\mathbf{H}_i \leftarrow 0$
  for b **in** $\{1, \ldots, n\}$ **do**
    $p \leftarrow \mathbf{M}_{i,b}$/*rowsum*
    $\mathbf{H}_i$ −= $p \log_2 p$
(optionally call *squeeze*)
item-order ← sort items $i \in \{1, \ldots, m\}$ by $\mathbf{H}_i$ *(ascending)*
for b **in** $\{1, \ldots, n\}$ **do**
  $\mathbf{S}_b \leftarrow \sum_{i=1}^{m} \mathbf{M}_{i,b}$
bin-order ← sort bins $b \in \{1, \ldots, n\}$ by $\mathbf{S}_b$ *(descending)*
for *item* **in** *item-order* **do**
  for *bin* **in** *bin-order* **do**
    if $\mathbf{I}_{item} \leq \mathbf{C}_{bin}$ for all d dimensions **then**
      $\mathbf{P}[item] \leftarrow bin$
      $\mathbf{C}_{bin}$ −= $\mathbf{I}_{item}$

We explain our *MonteCarlo* technique through an example. A formal description is given in Algorithm 1. Suppose that we have five items ($m = 5$) to place in four bins ($n = 4$) considering two resources ($d = 2$). We can think of the resources as the number of vCPU cores and GB of RAM, or any other arbitrary resources. As listed at the left of Figure 1, our items' resource demands are $(1, 2)$, $(4, 4)$, $(4, 1)$, $(1, 1)$, and $(4, 8)$. Our bins' resource capacities are $(1, 1)$, $(8, 12)$, $(4, 4)$, and $(2, 8)$. This example is constructed so that the last and most demanding item $(4, 8)$ can only fit into the "largest" bin $(8, 12)$, the least demanding item $(1, 1)$ can fit anywhere, and only it can fit into the "smallest" bin $(1, 1)$. The other items have a variety of options as to where they can be placed.

		Bins				
		0	*1*	*2*	*3*	*Entropy*
Items		$(1, 1)$	$(8, 12)$	$(4, 4)$	$(2, 8)$	*H*
0	$(1, 2)$	0	**32**	**22**	**23**	1.564 bits
1	$(4, 4)$	0	**53**	**24**	0	0.895 bits
2	$(4, 1)$	0	**38**	**27**	0	0.979 bits
3	$(1, 1)$	**29**	**27**	**19**	**25**	1.983 bits
4	$(4, 8)$	0	**51**	0	0	0 bits
	Sum:	29	201	92	48	

Fig. 1: *MonteCarlo* matrix example with $k = 100$ samples

Starting with an $m \times n$ matrix of zeros, with one row for every item and one column for every bin, we apply a predetermined number k of Monte Carlo samples. The result of $k = 100$ samples is shown in Figure 1. For each sample, we randomly attempt to place each item into a random bin. First, we create a temporary copy of the state of bins. Then we place each item, in random order, into the first bin in which it will fit (searching bins in random order). After each placement of item i into bin b, we record it by incrementing value (i, b) in the matrix and subtracting the allocated resources from the availability of the bin's temporary state. If at any time an item cannot be placed in any bin, we simply skip it and continue to the next item. This naive randomized technique for each sample is not expected to yield feasible solutions in which every item is placed; many or all samples will have skipped some items. Notice in Fig. 1 that the sum of each *row* is not always equal to the number of samples (100). Many samples did not place every item. Each sample, however, produces a little bit of information, and the aggregation of these samples provides sufficient information for a well-informed FFD placement.

After populating the matrix with k random samples, we calculate the *Shannon entropy* of each *row* and the *summation* of each *column*, which are used to sort items and bins respectively for an item-centric first fit decreasing placement.

Each row can be thought of as a probability distribution of where an item might be placed, taking into consideration interactions with other items. Our "decreasing" of FFD is a *decreasing* order of information about these probability distributions, or *increasing* by entropy. In our example there are four bins, so the most information we can have is $\log_2 4 = 2$ bits. The item with the lowest entropy (uncertainty) of 0 bits, or greatest information of 2 bits, is the "largest" item $(4, 8)$.

Since there is no uncertainty from our sampling (the choice is obvious), we wish to place item 4 first. On the other hand, the "smallest" item $(1, 1)$ has the greatest entropy of 1.983 bits, which is nearly the maximum 2 bits. It can likely be placed in any bin and has the least restrictive demands, so we wish to place it last. We sort items by *increasing entropy*: $(4, 8)$, $(4, 4)$, $(4, 1)$, $(1, 2)$, $(1, 1)$, which is consistent with intuitive notions of decreasing resource demand.

We sort bins in decreasing order by *popularity*, or column sum from the matrix. In Fig. 1, the bin most frequently used by random sampling is bin 1. It has the largest resource capacities $(8, 12)$ and the greatest column sum 201. We use this sample popularity to predict the relative "size" of bins, and search bins in decreasing order by popularity. It is interesting to compare bins 2 $(4, 4)$ and 3 $(2, 8)$. Each has twice the capacity of the other in one dimension; however, bin 2 has almost twice the column sum (92) as bin 3 (48). This suggests, without normalization or direct comparison of resources, that having more of the first resource makes a bin "larger" than more of the second. The search order of bins is $(8, 12)$, $(4, 4)$, $(2, 8)$, and $(1, 1)$.

Once we obtain a ranking of items and bins, we apply an item-centric first fit decreasing placement. We place each item, in increasing order of entropy, in the first bin in which it will fit, searching through bins by increasing popularity. The time complexity of the proposed *MonteCarlo* heuristic, shown in Alg. 1, is $O(dnm+n \log n+m \log m)$ for constant samples k.

Using a comparison of entropies we determine the relative "restrictiveness" of an item's placement, the fundamental property of its "size." Notice that this technique does not require normalization of resource values. There is no addition or multiplication of incompatible units of measure and no direct comparison of one dimension to another.

The sorting of items by *MonteCarlo* relies on information accumulated about each item from random sampling. This requires a sufficient number of samples k such that the distribution of each row of the matrix has converged to a representative state. The entropy for all items can be represented as a column vector, as on the right side of Fig. 1. This example represents (by construction) an ideal case, where its mean of 1.084 bits is in the middle of the possible range $[0, 2)$ with a wide standard deviation of 0.672 bits.

In some cases, even with a sufficient number of random samples, there is too little information gained to provide a valuable item ordering, which can lead to poor performance. For example, suppose that we have only large bins and that all items can be placed anywhere. If there is not enough contention for the last available resources, then little signal appears in the entropy vector. In such cases, the entropy is high for all items and with low variance. We have observed this in cases with low resource utilization variance. In order to improve item ordering in such situations, we propose a *"squeeze"* method of additional modified random samples that artificially lower bin capacities for resource contention.

After populating the *MonteCarlo* matrix and calculating entropy for all items, if the standard deviation of this entropy vector is less than 10% of its mean, we apply additional "squeeze" samples. This threshold is experimental; when greater than 10%, we considered *MonteCarlo*'s performance to be adequate without using additional squeeze samples. For each additional sample, we "squeeze" each bin by lowering its capacity in all dimensions by 1% and apply a random placement as before. In order to boost the effect of these samples on the entropy calculation, we gradually increase the weight of the tallies applied to the matrix. In less than 100 samples of 1% reduction each, eventually the bins become so small that no items are placed, and the squeezing stops.

IV. Evaluation

In this section, we evaluate the performance of the proposed *MonteCarlo* heuristic by comparing it to existing techniques as well as optimal solutions.

A. Experimental setup

We performed simulations using an in-house simulator written in Python supported by IBM's CPLEX LP solver [21] to obtain optimal results. Realistic workloads vary widely across organizations, making it difficult to generalize from any given set of real workloads. Following the work of Panigrahy et al. [13], we ran the heuristics on synthetic instances for $m = n = 100$ and $m = n = 1000$ with a variety of resource utilization scenarios to examine the behavior of the proposed and existing techniques in greater detail. We repeated each experiment 10 times and present mean, median, and various percentiles. Due to space limitations, we share only our experimental results in detail for $m = n = 100$, and for a representative three cases from each resource utilization scenario. Extensive results for all *feasible* resource utilization variances can be accessed from the project web page [22].

1) Item demands: We model our item resource demands on Amazon Web Services (AWS) EC2 instance types [23]. Items are randomly sampled (with replacement) from compute optimized "C5" EC2 instance types with a diversity of resource requirements. We use three resources: virtual CPU cores, gigabytes of memory, and gigabits of network bandwidth. The distribution of these resource requirements is summarized in Table II.

Resource	Minimum	Median	Mean	Maximum
vCPUs	2 cores	36 cores	42 cores	96 cores
Memory	4 GB	72 GB	84 GB	192 GB
Network	10 Gbps	10 Gbps	15.2 Gbps	25 Gbps

TABLE II: Distribution of item (VM) resource requirements, taken from AWS EC2 "C5" compute instances [23].

2) Bin capacities: For VMP, the collection of bins models the state of available resources in the cloud infrastructure at the moment of assignment. The available computing resources may not be proportional to the collective demand of the VMs. Some resources may be in higher demand relative to their availability than others, leading to a bottleneck in one or more resources. In order to evaluate heuristic performance

under a variety of such resource bottlenecks, we control the distribution of bin capacities to provide a predetermined *resource utilization ratio* $\mathbf{R}$, which is a vector representing total item demand divided by total bin capacity for each resource dimension such that $\mathbf{R} = \sum_i \mathbf{I}_i / \sum_b \mathbf{C}_b = m\mu_i / n\mu_b$. Bin capacities are sampled from a uniform distribution with mean capacity vector μ_b that satisfies this ratio.

We evaluate using three resource utilization conditions:

Case 1: Low — low resource utilization variance between individual resource dimensions, where the *maximum* difference between any two dimensions d of the demand ratio $\mathbf{R}_d$ is 10% or less.

Case 2: Medium — medium resource utilization variance, where the *maximum* difference between any two dimensions d of the demand ratio $\mathbf{R}_d$ is greater than 10% and less than 30%.

Case 3: High — high resource utilization variance, where the *maximum* difference between any two dimensions d of the demand ratio $\mathbf{R}_d$ is at least 30%.

B. Heuristics

In this section, we summarize the heuristics used in our evaluation. Since some of the previous heuristics are designed for VBP with homogeneous bins, we generalize them to the heterogeneous bins of VBPHB for fair comparison. In this generalization, we sort bins with a similar technique used to sort items. In addition, we use normalization in item sizes and bin capacities since it improves the performance of certain heuristics. We implemented the following heuristics for evaluation:

Lex — Lexicographical heuristic as described by Kou et al. [11], where items are sorted in decreasing order based on the lexicographical comparison of each dimension. For heterogeneous bins, we similarly generalize bin sorting to decreasing order based on the lexicographical comparison of bin capacities.

MaxComp — Maximum Component heuristic as described by Kou et al. [11], where items are sorted in decreasing order based on the resource dimension with the maximum value. For heterogeneous bins, we similarly generalize bin sorting to decreasing order based on the bin capacity dimension with the maximum value.

MaxSum — Maximum Sum heuristic as described by Kou et al. [11], where items are sorted in decreasing order based on the sum of all resource dimensions. For heterogeneous bins, we generalize bin sorting to decreasing order based on the sum of all bin capacity dimensions.

Prod — Product heuristic as described by Maruyama [17], where items are sorted in decreasing order by the product of individual resource dimensions. For heterogeneous bins, we similarly generalize bin sorting to decreasing order of the product of bin capacity dimensions.

AvgSum — Average sum heuristic applied to VMP by Panigrahy et al. [13]. Items are sorted in decreasing order by the dot product of each item's demand and the average resource demand (*avgdem*). For heterogeneous bins, we similarly generalize bin sorting to decreasing order of the dot product of each bin's capacity and *avgdem*. We also tested the exponential version of this heuristic [13]; however, we do not present its results as its performance was similar to *AvgSum*.

DotProd — Dot product technique as described and originally proposed by Panigrahy et al. [13], where item ordering is performed based on a weighted dot product of the item demand and remaining bin capacity vectors. For heterogeneous bins, we generalize bin ordering as the dot product with *avgdem*.

DP (Gabay) — Gabay's generalization [14] of Panigrahy's *DotProd* for heterogeneous bins. For bin sorting, it presents three variants for weight vectors: i.) no weight, ii.) weight based on bin capacity, and iii.) weight based on bin capacity and item demand. Our evaluation indicates that the weight based on only bin capacity (ii) yields the best results, so we only present this variant.

AR — Aggregated Rank from Jangiti et al. [20], where items are first ranked for each dimension separately, and then aggregate rankings are calculated by summing individual rankings for each dimension [20]. We generalize this concept to heterogeneous bins by sorting bins based on a similar aggregated ranking of bin capacities.

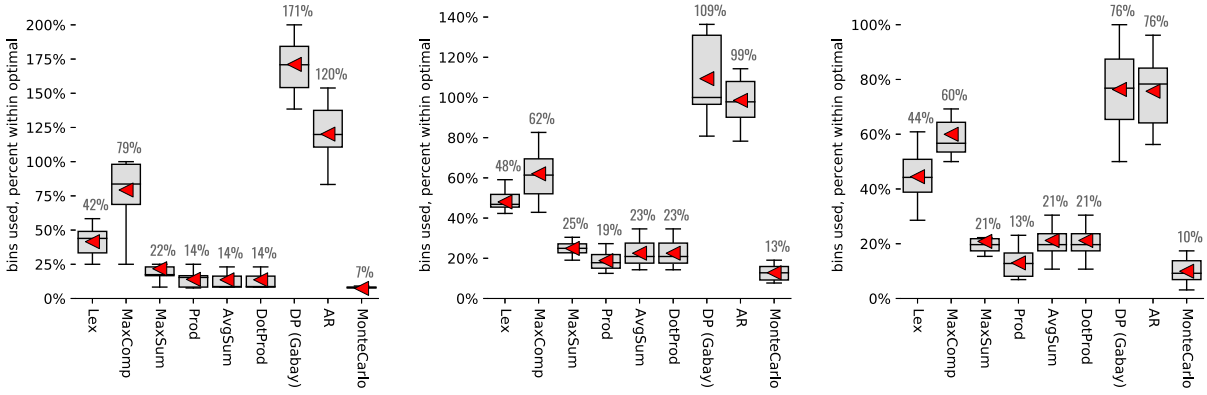
MonteCarlo — Our proposed heuristic, as described in Sec. III. We run with and without the presented squeeze technique and use the solution with the fewer number of bins.

Optimal — The optimal solution, as described by the integer linear programming formulation in Sec. II-B, solved using IBM CPLEX [21]. Due to its high execution time, we only obtain the optimal result for the $n = m = 100$ problem size.

C. Experimental results

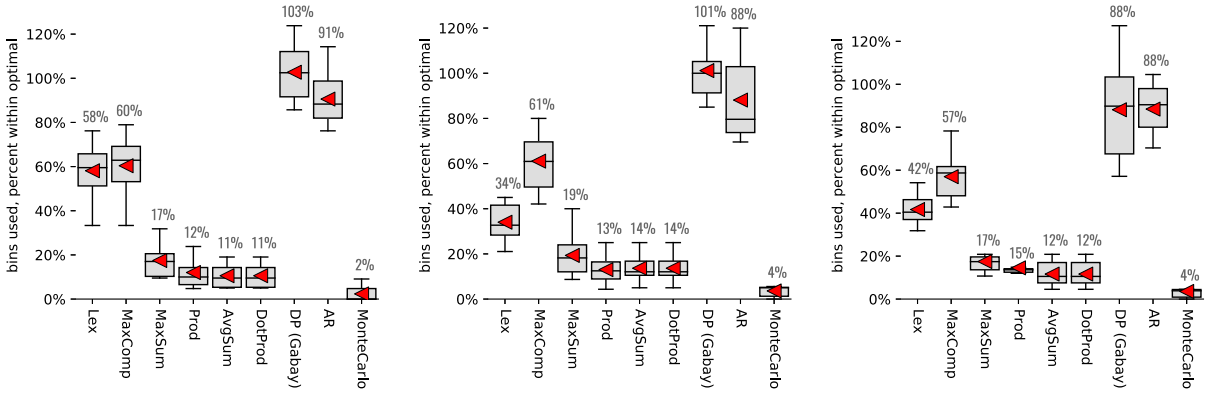
Figures 2, 3, and 4 show the performance of the heuristics compared to the optimal values using $m = n = 100$, for low (Case 1), medium (Case 2), and high (Case 3) resource utilization variances, respectively. For each resource utilization variance, we selected three representative ratios shown with three digits, where the first digit represents the CPU utilization, the second digit represents the memory utilization, and the third digit represents the network utilization in multiples of 10%. For example, ratio 211 represents a scenario from Case 1 (low resource utilization variance), where the total requested CPU cores is 20% utilization of the available data center, memory utilization is 10%, and network utilization is 10%. Extensive results including all *feasible* resource utilization ratios as well as the $m = n = 1000$ problem size are provided in the project website [22]. Infeasible ratios are discarded as no heuristics, including the optimal technique, could find any solution due to high resource contention.

In all figures, the mean is marked with a red triangle and written above. Each distribution is represented by a standard box and whisker plot; the bar in the middle of the box is



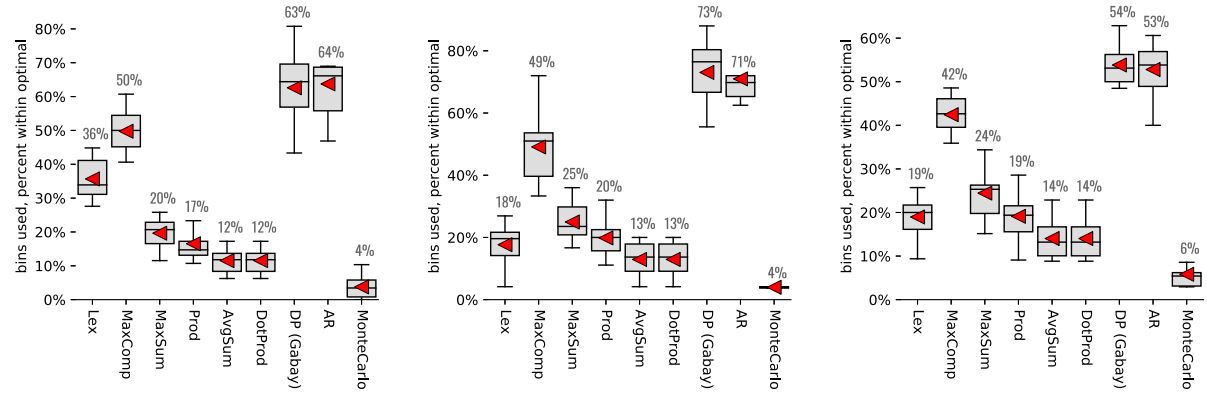
(a) Resource utilization ratio: 211 (b) Resource utilization ratio: 323 (c) Resource utilization ratio: 333

Fig. 2: Performance within optimal for case 1: *low* resource utilization variance, $n = m = 100$



(a) Resource utilization ratio: 231 (b) Resource utilization ratio: 321 (c) Resource utilization ratio: 331

Fig. 3: Performance within optimal for case 2: *medium* resource utilization variance, $n = m = 100$



(a) Resource utilization ratio: 341 (b) Resource utilization ratio: 411 (c) Resource utilization ratio: 521

Fig. 4: Performance within optimal for case 3: *high* resource utilization variance, $n = m = 100$

the median, box ends are Q1 (25th percentile) and Q3 (75th percentile), and whiskers are the farthest points within 1.5 IQR (Q3 − Q1) from the box.

For all three resource utilization cases, the proposed *MonteCarlo* heuristic outperforms the other heuristics and achieves results that are close to optimal. The performance of *MonteCarlo* is not affected by the resource utilization variance, while the other heuristics' performances vary wildly depending on whether their assumptions are held by the system's resource utilization. For instance, *Prod* and *MaxSum* assume low resource utilization variance as they simply perform sorting based on the product or summation of individual resource

dimensions, and therefore perform closer to *MonteCarlo* for Case 1, shown in Figure 2. Actually, Case 1 represents an "easy" scenario for many heuristics as they simply assume even resource utilization. However, the performance of other heuristics degrade compared to *MonteCarlo* in Figure 3 for Case 2 with medium variance, and in Figure 4 for Case 3 with high variance. In 21% of these cases, *MonteCarlo* found the optimal result, and its performance was within 10% of optimal for Case 1, within 3.1% of optimal for Case 2, and within 4.5% of optimal for Case 3, on average. For medium and high variance scenarios, *MonteCarlo* found a solution within 2-5% of optimal.

	Mean within optimal	**Mean bins used**	
Heuristic	100 bins total	100 total	1000 total
Lex	51.7%	40.4	406.4
MaxComp	52.5%	40.6	409.6
MaxSum	24.4%	34.1	342.6
Prod	20.7%	33.2	333.1
AvgSum	17.2%	32.1	321.3
DotProd	17.2%	32.1	321.3
DP (Gabay)	83.3%	46.6	465.4
AR	77.4%	45.8	469.1
MonteCarlo	9.8%	30.6	301.0
Optimal	—	27.5	—

TABLE III: Number of bins used, aggregation of all ratios

Table III gives an aggregate comparison for all resource utilization ratios. The LP formulation for the optimal solution is solved only for the smaller $n = m = 100$ bin problem size due to its prohibitive running time. In the first column, we compare the number of bins used by each heuristic against the optimal solution for each problem instance solved (given as a percentage worse than optimal), then average across all ratios. The last two columns give the average of the number of bins used across all ratios, without comparison to the optimal solution. Notice how the number of bins used scales consistently to the larger ($n = m = 1000$) problem size. For all possible *feasible* ratios (including the ones not presented due to space limitations, but available on the project website [22]), the performance of *MonteCarlo* was within 10% of the optimal solution on average.

V. Conclusion

In this paper, we formally defined and formulated the server consolidation in cloud data centers as an optimization problem and solved it using linear programming techniques. Next, we proposed a low cost heuristic for effective server consolidation using a Monte Carlo method and Shannon entropy. In our evaluation, the proposed heuristic outperformed existing methods by achieving within 2-5% of the optimal solution when the resource utilization variance was medium to high, and within 10% of the optimal on average for all resource utilization scenarios. Therefore, we believe that the proposed heuristic can provide a tremendous value towards energy efficient cloud data centers. Our future work includes investigating the ways of dynamically improving the information gain of the proposed heuristic under the scenarios where the gain is poor.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under grants CNS-1657296, CNS-1828521, CNS-1801513, and OIA-1849213.

References

[1] P. Bertoldi, M. Avgerinou, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," JRC, European Union, Tech. Rep., 2017, https://core.ac.uk/download/pdf/141667150.pdf.

[2] *Electricty Consumption of Countries*, CIA World Factbook, https://www.cia.gov/library/publications/the-world-factbook/rankorder/2233rank.html.

[3] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering Cloud Computing: Foundations and Applications Programming*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[4] A. Beloglazov, R. Buyya, Y. C. Lee *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in Computers*, vol. 82, pp. 47–111, 2011.

[5] A. Shehabi, S. Smith, D. Sartor *et al.*, "United states data center energy usage report," 6 2016, https://www.osti.gov/servlets/purl/1372902/.

[6] V. V. Vazirani, *Approximation Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2001.

[7] M. R. Garey, R. L. Graham, and J. D. Ullman, "Worst-case analysis of memory allocation algorithms," in *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '72, 1972, pp. 143–150.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman, 1979.

[9] D. Friesen and M. Langston, "Variable sized bin packing," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 222–230, 1986.

[10] M. Garey, R. Graham, D. Johnson *et al.*, "Resource constrained scheduling as generalized bin packing," *Journal of Combinatorial Theory, Series A*, vol. 21, no. 3, pp. 257 – 298, 1976.

[11] L. T. Kou and G. Markowsky, "Multidimensional bin packing algorithms," *IBM J. Res. Dev.*, vol. 21, no. 5, pp. 443–448, sep 1977.

[12] C. S. Rao, J. J. Geevarghese, and K. Rajan, "Improved approximation bounds for vector bin packing," *CoRR*, vol. abs/1007.1345, 2010.

[13] R. Panigrahy, K. Talwar, L. Uyeda *et al.*, "Heuristics for vector bin packing," January 2011. https://www.microsoft.com/en-us/research/publication/heuristics-for-vector-bin-packing/

[14] M. Gabay and S. Zaourar, "Vector bin packing with heterogeneous bins: application to the machine reassignment problem," *Annals of Operations Research*, vol. 242, no. 1, pp. 161–194, Jul 2016.

[15] N. Bila, E. J. Wright, E. D. Lara *et al.*, "Energy-oriented partial desktop virtual machine migration," *ACM Trans. Comput. Syst.*, vol. 33, no. 1, pp. 2:1–2:51, Mar. 2015.

[16] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, vol. 40, no. 4, pp. 85–103, 1972.

[17] K. Maruyama, S. K. Chang, and D. T. Tang, "A general packing algorithm for multidimensional resource requirements," *International Journal of Computer & Information Sciences*, vol. 6, no. 2, pp. 131–149, Jun 1977.

[18] M. Stillwell, D. Schanzenbach, F. Vivien *et al.*, "Resource allocation algorithms for virtualized service hosting platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.

[19] P. Silva, C. Perez, and F. Desprez, "Efficient heuristics for placing large-scale distributed applications on multiple clouds," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 483–492.

[20] S. Jangiti, E. Sri Ram, and V. S. Shankar Sriram, "Aggregated rank in first-fit-decreasing for green cloud computing," in *Cognitive Informatics and Soft Computing*. Singapore: Springer, 2019, pp. 545–555.

[21] I. CPLEX, "IBM ILOG CPLEX optimization studio for academics: High-performance software for mathematical programming and optimization," http://www.ilog.com/products/cplex/.

[22] *Project Webpage*, Computer Systems Laboratory (CSL) at the University of Louisville, http://cecs.louisville.edu/csl/mc.

[23] *Amazon EC2 VM Instance Types*, Amazon, 2019, https://aws.amazon.com/ec2/instance-types/.