A Fair Comparison of VM Placement Heuristics and a More Effective Solution

Xi Li, Anthony Ventresque, John Murphy
Lero and School of Computer Science and Informatics
University College Dublin, Dublin, Ireland
Email: xi.li@ucdconnect.ie; {anthony.ventresque, j.murphy}@ucd.ie

James Thorburn
IBM Software Group, Toronto, Canada
Email: jthorbur@ca.ibm.com

***Abstract*—Data center optimization, mainly through virtual machine (VM) placement, has received considerable attention in the past years. A lot of heuristics have been proposed to give quick and reasonably good solutions to this problem. However it is difficult to compare them as they use different datasets, while the distribution of resources in the datasets has a big impact on the results. In this paper we propose the first benchmark for VM placement heuristics and we define a novel heuristic. Our benchmark is inspired from a real data center and explores different possible demographics of data centers, which makes it suitable when comparing the behaviour of heuristics. Our new algorithm, *RBP*, outperforms the state-of-the-art heuristics and provides close to optimal results quickly.**

***Index Terms*—VM Placement; Server Consolidation; Heuristic; Benchmark**

I. Introduction

With the rapid growth of computing resources' demand, the scale of the IT infrastratures in data centers increases sharply, with workloads distributed across the centers. However, the average server utilization in many large organizations is at or below 20%-30% [1]. In order to drive higher utilization, server consolidation [2], [3] has been widely adopted to locate workloads on fewer, powerful physical machines (PMs, also referred to as hosts or servers in the remainder of this paper) to enable the decommissioning of the unneeded PMs. With virtualization technology, workloads are encapsulated in virtual environments which are referred to as virtual machines (VMs) that provide application isolation when co-located on one PM and can be migrated easily. The key of consolidation becomes deciding which VM should be placed on which PM. This underlying VM placement problem is usually considered as an instance of the NP-hard bin-packing problem (BPP), which makes the finding of an optimal solution a challenge in terms of quality and time, given that the problem consists of tens of thousands of PMs and hundreds of thousands of VMs. Quality (e.g., number of PMs required and resource wastage [4]) and time to find a good solution are both important as the first one allows the solutions to save money and the second is essential for capital allocators to make their decisions (see [5]).

Many heuristics have been proposed to solve this problem [6]–[8], and we propose a novel effective one in this paper (see Section III): *RBP*, a *R*esource *B*alancing VM *P*lacement algorithm that aims to maintain a balanced resource utilization among different dimensions of one PM and minimize the number of PMs required for accommodating a certain set of VMs. In short, *RBP* fits in VMs that do not saturate one type of resource of a PM and waste the others.

However, very often papers in this area are based on their own datasets, either real ones or synthetic ones defined by researchers (sometimes using characteristics of real data centers). This makes a comparison between two algorithms presented in two papers challenging, for two reasons: (i) the demographics of data centers can vary and (ii) heuristics are dependent on the demographics of data centers.

Firstly, there is a large variety of possible PMs and VMs and the distribution of them in different data centers can be very different. For instance, one can be "historic" or acquired recently and hosts old machines that are not powerful in terms of CPU or RAM; while another one can be brand new with the latest version of powerful servers. Moreover, different types of workloads have distinct resource requirements (e.g., scientific application are compute-intensive, web servers require mostly network resources) and the proportion of a certain type of workloads is different in data centers or clusters with different purposes (a recent work shows that for clusters at Google [9]).

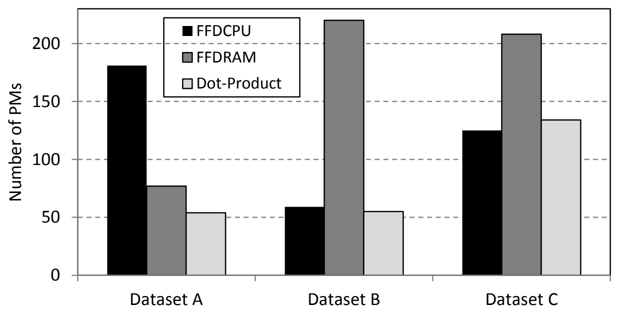


Fig. 1. Performance of 3 heuristics against various datasets.

Secondly, it is clear from a small example that heuristics behave differently against various datasets. We plotted in Figure 1 the results of three algorithms in terms of the number of PMs required: First-Fit Decreasing (FFD) according to CPU and RAM, and Dot-Product [6], on three datasets that are inspired from a real world data center's demographics (1,000 PMs and 2,000 VMs). For all datasets, the CPU and RAM demands of VMs are "normal", i.e., most of them have balanced CPU and RAM demands and are in medium size, while the CPU and RAM capacities of PMs vary: dataset A contains mainly large and small, with few medium servers which are either powerful in CPU or RAM, but are not well balanced in these two types of resources; dataset B is mostly composed of large servers with large RAM capacity and relatively less CPU; dataset C also consists of servers with large RAM and small CPU capacity, but are in small size. We can see that the three algorithms perform differently on the different datasets, which hints at telling us that when comparing heuristics for VM placement, one needs to try different demographics.

To overcome this problem, this paper presents a benchmark for the comparison of VM placement algorithms. This benchmark generates a large number of "virtual" data centers with different demographics, inspired from one original real data center. It combines different distributions of VMs and PMs that cover many possibilities of data center demographics.

In the rest of this paper, we present a definition of the VM placement problem (Section II), some related work and our proposed algorithm *RBP* (Section III), the benchmark (Section IV), and the evaluation of our algorithm's performance (Section V). There are two main contributions for this paper: the benchmark for VM placement algorithms evaluation, and RBP, our proposed heuristic that outperforms classical placement heuristics in terms of PMs required (9.9%-46.6% less) and resource wastage (21.2%-64.6% better), and produces results close to the optimization problem solver (CPLEX) [10]: only 5.54% more PMs needed on average while orders of magnitude faster.

II. Problem Definition

Every VM or PM can be defined as a vector (i.e., $\vec{v}$ or $\vec{p}$) in the d-dimensional space defined by the hardware resources, with each dimension representing one type of resource (e.g., CPU, RAM, network, disk): see Figure 2.

We denote by $V = \{\vec{v}_1, \dots \vec{v}_n\}$, with $n = \|V\|$, the set of VMs, and by $P = \{\vec{p}_1, \dots \vec{p}_m\}$, with $m = \|P\|$, the set of PMs. Many tools exist for monitoring and collecting hardware configuration information and VM resources usage, such as ITM (IBM Tivoli Monitoring) [11]. These tools can help fill in the vectors in V and P. PMs are defined by their hardware configuration while VMs' resource demands are estimated from their usage over time. Workload estimation and forecasting for VMs [12], [13] is a challenge and an important area of research but it is out of the scope of this paper and it has been shown that historical traces and non peak (i.e., $90th$ percentile or more) resource usage can produce good estimations [8], [14]–[16]. In this paper, we estimate each VM's resource demand based on the high percentile (i.e., $95th$) of its resource usage over the past 90 days. As the performance degradation caused by unusual peaks or interference between co-located VMs is often unavoidable, we apply the common strategy [6] to tolerate it, which allows VMs to use up to 75% of the total capacity of a PM, leaving 25% as a buffer for virtualization overhead or resource utilization spike (see Figure 2b).

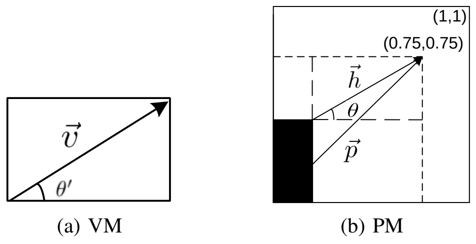


Fig. 2. Vectors and shapes of a VM's demand and a PM's capacity. (a) $\vec{v}$: vector of a VM's demands; θ' represents the shape of a VM's demands. (b) $\vec{h}$: vector of a PM's residual capacity; $\vec{p}$: vector of 75% of a PM's total capacity; θ represents the shape of a PM's residual capacity; The VM in (a) has similar shape as the residual capacity in (b).

The VM placement problem, consisting in placing VMs on to PMs, is usually treated as a BPP, considering VMs as items and PMs as bins.

Definition 1 (VM Placement): We denote any placement as a mapping $\mathcal{P} : V \mapsto P$, such that $\mathcal{P}(\vec{v_i}) \to \vec{p_j}$, and (i) $\forall \vec{v_i} \in V, \mathcal{P}(\vec{v_i}) = \vec{p_j} \wedge \mathcal{P}(\vec{v_i}) = \vec{p_k} \iff j = k$, i.e., every VM can be placed on only one PM, (ii) $\forall \vec{p_j}, \forall k \in [1 \dots d], \vec{p_j}[k] \geq \sum_{\{\vec{v_i} | \mathcal{P}(\vec{v_i}) \to \vec{p_j}\}} \vec{v_i}[k]$, i.e., the total resource demands of all VMs placed on $\vec{p_j}$ can not exceed $\vec{p_j}$'s capacity.

There are two costs associated with every VM placement $\mathcal{P}_k$: (i) the number of PMs used:

$$cost_{hosts}(\mathcal{P}_k) = |\{\vec{p_j} | \vec{p_j} \in P, \exists \vec{v_i} \in V, \mathcal{P}_k(\vec{v_i}) = \vec{p_j}\}|$$

and (ii) the resource wastage [4], which indicates how balanced the resources of different dimensions are occupied by measuring the sum of the differences between every dimension's normalized remaining resource and the smallest one:

$$cost_{waste} = \sum_{\{\vec{p_j} | \vec{p_j} \in P, \exists \vec{v_i} \in V, \mathcal{P}_k(\vec{v_i}) = \vec{p_j}\}} \Bigg(\alpha \sum_{d_i=1}^{d} \left(\vec{h}[d_i]/\vec{p_j}[d_i] - \min_{d_k=1}^{d} (\vec{h}[d_k])/\vec{p_j}[d_k] \right) \Bigg)$$

with $\vec{h}$ the vector of a PM's residual capacity and α the buffer threshold (i.e., $\alpha = 0.75$). Classically, heuristics for VM placement are evaluated on these two costs, or costs derived from them (e.g., power consumption), that they try to *minimize* [4], [17].

III. VM Placement Algorithms

In this section, we propose two new heuristics following some related work on the existing VM placement algorithms.

A. Algorithms in the Literature

The approaches for the VM placement problem include heuristics [4], approximation algorithms [12], and approaches using constraint programming models [18], [19], integer programming models [17], [20]. However, the majority of these studies concentrate on the optimality of the solutions, and are computation intensive that require long execution time.

Xu et al. [4] proposed a genetic algorithm with fuzzy multi-objective evaluation to simultaneously minimize the total resource wastage, power consumption and thermal dissipation costs. However, the hosts considered in their experiments are uniform, and the algorithm takes about 3 minutes to solve a problem with $1,000$ PMs and $2,000$ VMs. Feller et al. [17] proposed an Ant Colony Optimization (ACO) based algorithm that produces solutions close to that of CPLEX. However, the ACO based algorithm takes about 2 hours to find the optimal placement on 167 PMs for 600 VMs. These approaches can compute near optimal results in small-scale experiments, but may not be applicable in large-scale environments due to their long execution time.

Greedy heuristics are potential solutions when considering scalability, complexity and process time. Analytical and empirical results also suggest that FFD is the best heuristic for (one-dimensional) bin packing problem [21]. Sandpiper [8] presented a heuristic based on its metric *Volume*. However, it is designed for hotspot mitigation. The *Volume* metric does not fit in our case as it indicates servers' resource utilization level, not their capacity. Lee et al. [6] provided a review of various FFD-based (i.e., FFDRAM, FFDCPU, FFDProd and FFDSum) and dimension-aware heuristics (i.e., Dot-Product and l2norm). Srikantaiah et al. [7] proposed a heuristic making use of the sum of Euclidean distances. In this paper, we compare our proposed solutions with these seven commonly used heuristics. A brief introduction for these heuristics is provided below.

a) FFD-based: FFDRAM considers RAM as the dominant resource so that PMs and VMs are ordered by their RAM capacity and RAM demands respectively in descending order. The other FFD algorithms work similarly using different sorting metrics, namely CPU, the product of resources and the weighted sum of resources. The weight is calculated as the ratio between the total demand of one type of resource and the capacity of that resource on the host.

b) Dot-Product: It places the VM that maximizes the dot product $\sum_r^d w_r \vec{v}[r]\vec{h}[r]$, where $\vec{h}[r]$ is the host's residual capacity of resource r, $\vec{v}[r]$ is the VM's demand of resource r, and w_r is the weight of resource r calculated as it is in FFDSum.

c) l2norm: l2norm looks at the difference between the vector $\vec{v}$ and $\vec{h}$ under l^2 norm distance metric, and places the VM that minimizes the metric $\sum_r^d w_r(\vec{v}[r] - \vec{h}[r])^2$.

d) Sum of Euclidean distance: This algorithm tries to maximizes the sum of Euclidean distances of the current allocations to the optimal point at each server.

B. Resource Balancing Placement (RBP)

The idea behind our algorithm is that balancing the utilization of different resources on the target PMs makes the placement less prone to waste any resources and hence require fewer PMs. What *RBP* wants to avoid is the situation where only one type of resource on a host is fully occupied while plenty of other resources are wasted, then no more VMs can be placed on this host. In order to fulfill the above objectives, the shapes of the residual resources on hosts ($\vec{h}$) and the shapes of the resource requirements of VMs ($\vec{v}$) are considered.

We propose to use the angle between the vector of resource demands or residual capacity and the unit vector on one dimension to represent the shape of a VM or the residual capacity of a host. In two-dimension cases, the shape is hence represented as the angle between the demands/residual capacity vector and one dimension (θ' and θ in Figure 2). As the shapes of VMs in heterogeneous environments vary a lot, there is no standard shape that can suit more VMs than the other. Therefore, the best way to increase the possibility of hosting more VMs is to keep the normalized value of each dimension on the candidate host as close as possible. This can be achieved by always selecting the VM whose resource requirements are complementary to the occupied resources of the host. For instance, a VM requiring a large amount of CPU and small amount of RAM should be chosen when the considered host's remaining CPU is greater than RAM and vice versa. In other words, the VM whose shape is the most similar to the shape of the residual resources of the considered host should be placed on that host.

To this end, *RBP* places the VM that has the most similar shape with that of the currently considered PM's residual capacity, using the cosine similarity between $\vec{v}$ and $\vec{h}$. The bigger the value of Equation (1) the more similar the two shapes are.

$$\cos(\vec{v}, \vec{h}) = \frac{\sum_{r=1}^d \vec{v}[r] \times \vec{h}[r]}{\sqrt{\sum_{r=1}^d (\vec{v}[r])^2} \times \sqrt{\sum_{r=1}^d (\vec{h}[r])^2}} \tag{1}$$

The proposed algorithm *RBP* is presented in Algorithm 1. Considering that it might be difficult for some exceptionally big VMs to find a big enough host after other VMs are placed, we first sort VMs according to their sizes, which is represented as $\prod_{r=1}^d \vec{v}[r]$, and place the first VM of the list on the current host, which is the biggest one among available hosts (lines 4-5). If there is no host that the VM can fit in, there is no need to check the cosine similarity (lines 6-8). Otherwise, we sort VMs by their values of cosine and place the VM that has the maximum value.

Note that Dot-product contains cosine as $\vec{v} \cdot \vec{h} = \parallel \vec{v} \parallel \parallel \vec{h} \parallel \cos(\vec{v}, \vec{h})$. However, the lengths of the vectors have significant impact on the result of Dot-product. For instance, a VM with a small cosine, which means it is not complementary to the occupied resources of the host, can be selected if it has a big value of $\parallel \vec{v} \parallel$. We argue that the shape of a VM is more important to fully utilize resources in all dimensions.

C. RBP with Cosize (RBP$_c$)

An alternative to the cosine used in *RBP* is to take into consideration both the angle and the length of the VMs, as one could think that when two VMs have similar angles with the remaining capacity of a PM, the bigger one should be considered first. We call this modification of the cosine *cosize*,

Algorithm 1 VM Placement Algorithm - *RBP*

```
1: $pmList \leftarrow sortPMs$ //sort by available capacity
2: $placementSolution \leftarrow NULL$
3: for all $pm$ in $pmList$ do
4:     $vmList \leftarrow sortVMs()$ //sort VMs by size in descending order
5:     place the first $vm$ in $vmList$ that can fit in $pm$
6:     if no $vm$ can fit into the current $pm$ then
7:         continue //check next PM
8:     end if
9:     $vmList \leftarrow sortVMs(cosine)$ //sort VMs again by cosine similarity in descending order
10:    while $pm$ has available resources and $index < vmList.size()$ do
11:        $vm \leftarrow getVM(index)$
12:        if available resources on $pm$ are enough for $vm$ then
13:            $placementSolution.add(pm, vm)$
14:            $vmList.remove(vm)$
15:            $vmList \leftarrow sortVMs(cosine)$
16:            $index \leftarrow 0$
17:        else
18:            $index$++
19:        end if
20:    end while
21:    if $vmList = NULL$ then
22:        break
23:    end if
24: end for
25: return $placementSolution$
```

and we define it as:

$$cosize = \cos^{\alpha}(\vec{v}, \vec{h}) \times \prod_{r=1}^{d} \vec{v}[r] \qquad (2)$$

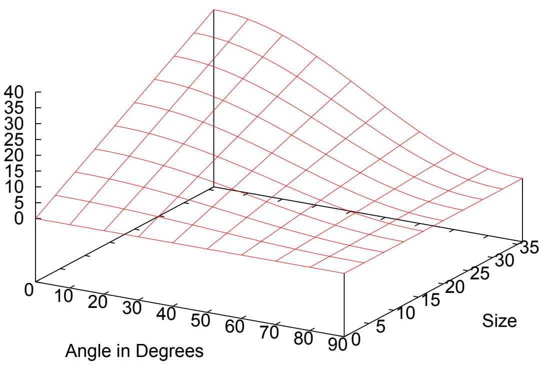
where α is used to adjust the significance of cosine similarity. Figure 3 shows the values of *cosize* with two different α and we can see that the bigger the α, the less impact the size has.

We call *RBP*$_c$ the modification of *RBP* using *cosize* as a similarity measure instead of cosine.

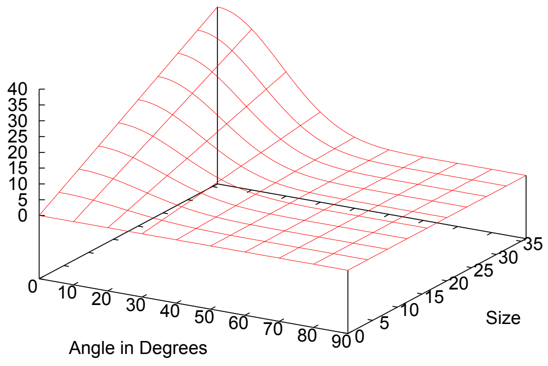
IV. Benchmark for VM Placement Algorithms Evaluation

Comparisons between placement algorithms in the literature often use real world traces from companies or synthetic data generated in an ad-hoc manner. Both solutions are to some extent biased as they describe a specific demographic of a data center: either a real one with its history and decisions made by previous managers (e.g., lots of new machines or old machines, of certain types) or an ideal single data center which may never be found in the real world. The problem is that this may affect the placement results, and the generality of the conclusions drawn upon them.

In our work, we collected real world traces from our industrial partner IBM. Based on that, we generated a *VM placement benchmark* that can represent a large variety of possible



(a) $\alpha = 2$



(b) $\alpha = 10$

Fig. 3. Cosize for VMs with Different Shape and Size

data centers with diverse demographics. The objective is to perform a more fair and general comparison between different placement algorithms, as the specificities of a particular dataset are eliminated.

Our benchmark is composed of 625 datasets, representing what we think is a comprehensive sample of what data centers' demographics can be. Each dataset consists of a set of VMs and a set of PMs, each of which has a specific distribution, and represents a potential data center with $2,000$ VMs and $1,000$ PMs. In the benchmark, we consider only two resources (CPU and RAM), which is coherent with most of the work in the literature – although our approach is not limited to these two resources. As we have already said (see Section II) VMs and PMs are vectors in a d-dimensional (here, $d = 2$) space and as such, they have an angle (e.g., θ' in Figure 2a) and a length (norm of vectors). We consider those as the crucial elements of the placement problem as the shape (angle) and the size (length) of the VMs and PMs make the placement more or less difficult, as it does for items and bins in the bin-packing problem [22].

We identified 5 different distributions for the angles and lengths of VMs or PMs in a data center (see Figure 4): i) normal, with the majority of VMs or PMs having an average angle/length; ii) U-shaped, with the extreme values of angle/length being more representative; iii) uniform, where all possible values of angle/length have the same density; iv) J-shaped, when high angle/length values are more likely; v)

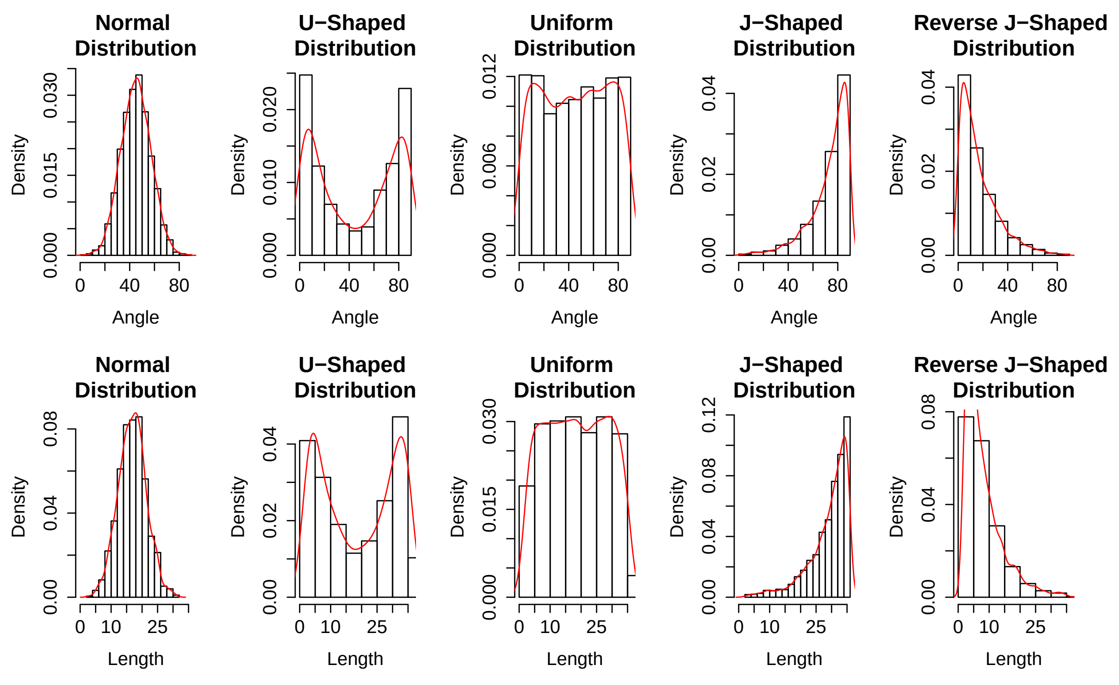


Fig. 4. Distributions of Angles and Lengths

reverse J-shaped, for distributions where small angles/lengths are more likely. The ranges of angles for VMs and PMs are the same (i.e., 0-90), while the ranges of lengths are different as PMs are generally bigger.

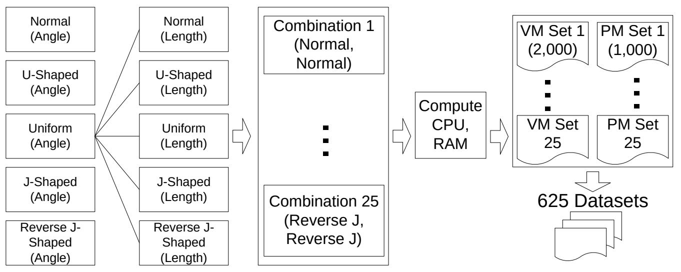


Fig. 5. Benchmark Generation

Figure 5 illustrates the generation of the benchmark. Each angle distribution can be combined with any one of the 5 length distributions. This generates 25 combinations (indexed as in Table I) for VMs and PMs respectively (i.e., 25 VM sets and 25 PM sets), or 625 couples of (VMs, PMs) demographics.

To generate VMs and PMs from (decoupled) distributions of angles and lengths, we use the following formulas:

$$RAM = length \times \cos(angle)$$

$$CPU = length \times \sin(angle)$$

Note that each point in a length or angle distribution together with one point in an angle or length distribution can construct an element of a VM or PM set, i.e., a VM or a PM. When creating a single VM or PM, each point of the angle distribution can be matched with a point in the length distribution. This gives us a huge number of possible VM sets or PM sets: $2,000!$ and $1,000!$ respectively for one angle and one length distribution (reminder: we want $2,000$ VMs and $1,000$ PMs). Obviously an exhaustive generation of datasets would be extremely expensive, and it is not clearly adding anything from the following test we performed. We generated pseudo randomly $2,000$ VM sets on one combination of angle and length distribution and compared the results of the various heuristics mentioned before against them. It appears that, for each heuristic, the coefficient of variation among the number of PMs required by each of the $2,000$ datasets is negligible (below 0.46%), which indicates that generating all the possible VM sets and PM sets from a combination of one length and one angle distribution is not necessary - we then pick a random one.

TABLE I
COMBINATIONS OF SHAPE/SIZE DISTRIBUTIONS

Shape (Angle) / Size (Length)	Normal	U-Shaped	Uniform	J-Shaped	Reverse J-Shaped
Normal	1	2	3	4	5
U-Shaped	6	7	8	9	10
Uniform	11	12	13	14	15
J-Shaped	16	17	18	19	20
Reverse J-Shaped	21	22	23	24	25

V. EXPERIMENTAL RESULTS

There are three things that we show in this section:

- that *RBP* is a better VM placement algorithm than the other state-of-the-art heuristics: for this we use the benchmark described in the previous section, and real traces from IBM;
- that *RBP*'s results are close to one of the best optimization solver, CPLEX: here we use a subset of our benchmark and real traces from IBM;
- that *RBP* gives its solutions faster than CPLEX, which is relatively slow: we extrapolate a scale-up experiment from IBM's traces.

A. Experimental setup

All algorithms were implemented in Java, and the experiments are carried out on a laptop with 8GB RAM and an Intel Core i7-2760QM 2.4GHz CPU.

Throughout the experiments section we use three metrics: (i) the number of PMs required by an algorithm to place all the VMs (see definition 1, Section II); (ii) the resource wastage [4], which indicates how balanced the resources of two dimensions are (reminder: we consider only CPU and RAM) and is described in definition 1 as well; (iii) the time to compute the VM placement solution.

The comparison between *RBP* and the various heuristics are done through 626 experiments using the benchmark we describe in Section IV (625 datasets) and one big real industrial dataset containing $1,327$ PMs and $8,686$ VMs. When we compare the quality of *RBP* to IBM CPLEX, we perform 26 experiments using (i) the 25 VM sets of the benchmark and the PM set from the above big IBM dataset, which contains $1,327$ PMs; and (ii) a small IBM dataset containing 342 PMs and $2,869$ VMs. The reason for using the big PM set with the 25 VM distributions is to guarantee enough PMs for all VMs of any distribution. However, a smaller IBM dataset is also used because CPLEX is computationally expensive and we could not run it on the benchmark, or the big VM sets from IBM for time and memory related reasons.

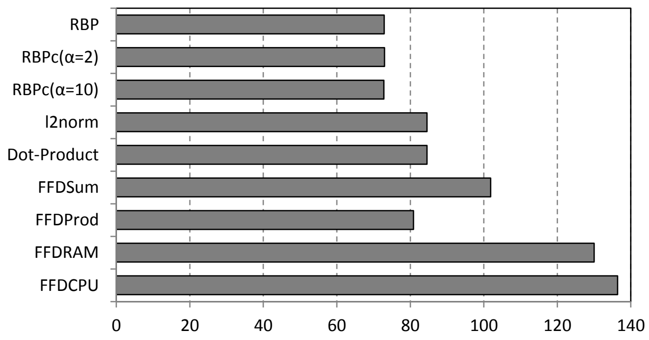
B. Comparison between VM placement heuristics

The average results from the 626 experiments are illustrated in Figure 6. The results show that RBP_c does not show an advantage over *RBP*, and *RBP* outperforms the other algorithms in terms of number of hosts required (Figure 6a) with 9.9%-46.6% reduction, and resource wastage caused (Figure 6b) with $21.2\%-64.6\%$ reduction. Another interesting observation is that the number of hosts follows the same trend as the resource wastage, certainly because a good placement (limiting the resource wastage) allows to put more VMs on the hosts and limits the number of hosts needed.

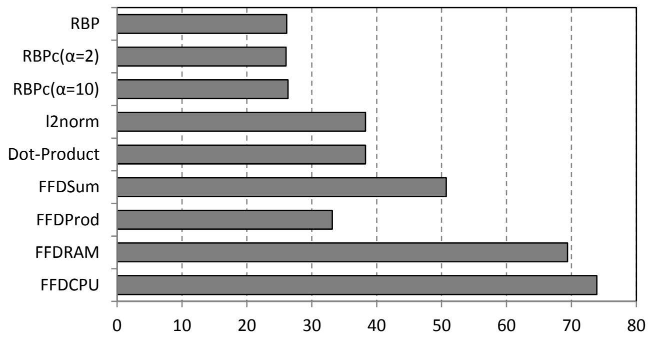
Dot-Product and l2norm have the same results in both measurements. Note that l2norm's metric can be transformed to $\sum_r^d(w_r(\vec{v}[r]^2+\vec{h}[r]^2)-2w_r\vec{v}[r]\vec{h}[r])$, where Dot-Product is $\sum_r^d w_r\vec{v}[r]\vec{h}[r]$. The different part is $w_r(\vec{v}[r]^2+\vec{h}[r]^2)$, which is a constant for a given pair of VM and PM. Therefore, it is not surprising that these two algorithm perform the same.

The result of the Sum of Euclidean distance is not presented because it requires extremely large number of hosts (i.e., 489.4 on average). The reason is that the Sum of Euclidean distance is a VM-centric on-line algorithm that deals with one new arriving VM at a time, considering on which one of the active servers the VM should be placed. It is not suitable for the scenario we are addressing. If we modified it to be PM-centric, which processes PMs one by one, it can be simplified to calculate the Euclidean distance of one PM after placing one VM on it instead of the sum of Euclidean distance of all PMs, since the Euclidean distance of the other PMs will remain the same as no VM is placed on them. However, the Euclidean distance of one host is $d(\vec{v},\vec{h}) = \sqrt{\sum_r^d(\vec{v}[r]-\vec{h}[r])^2}$, which is actually the square root of l2norm. Therefore, we did not implement the algorithm using the Euclidean distance of one host only.

We discovered in the experiments that the results of different algorithms vary when executed using different datasets, which suggests that the dataset distribution indeed has an impact on the results. The fluctuation is particularly big for FFDCPU and FFDRAM as they consider only one dimension and are vulnerable to the change of the shape and size of workload and capacity in a dataset. This observation would need a more complete study of the impact of the data center demographics on the VM placement heuristics. Due to space limitations, we present only the Min/Max/Avg results (i.e., number of PMs required) for the 25 experiments of each PM set (each PM set is combined with 25 VM set) in table II with the best result of each column in bold and the worst results in italic. The PM sets are indexed as the distribution combinations (table I). The corresponding VM sets' indexes are indicated as the subscript



(a) Number of Hosts



(b) Resource Wastage

Fig. 6. Comparison of Algorithms

TABLE II
RESULTS OF SEVERAL OF THE HEURISTICS AGAINST ALL THE PM DISTRIBUTIONS: THE FIRST NUMBER IS THE MINIMAL NUMBER OF HOSTS REQUIRED (THE CORRESPONDING VM DISTRIBUTIONS ARE INDICATED IN THE PARENTHESES), THE SECOND NUMBER IS THE MAXIMUM NUMBER OF HOSTS REQUIRED (THE CORRESPONDING VM DISTRIBUTIONS ARE INDICATED IN THE PARENTHESES), AND THE THIRD THE AVERAGE. BEST VALUES OF EACH COLUMN ARE IN **BOLD**, WHILE WORST ARE IN *italic*.

	PM Set 1 min:max [average]	PM Set 2 min:max [average]	PM Set 3 min:max [average]	PM Set 4 min:max [average]	PM Set 5 min:max [average]
FFDCPU	$26_{(24)}:\mathit{161_{(20)}}\ [\mathit{79}]$	$46_{(24)}:\mathit{385_{(20)}}\ [191]$	$\mathit{35_{(24)}}:\mathit{273_{(20)}}\ [\mathit{134}]$	$\mathit{52_{(24)}}:\mathit{472_{(20)}}\ [\mathit{220}]$	$34_{(24)}:\mathbf{184_{(19)}}\ [91]$
FFDRAM	$26_{(25)}:\mathit{161_{(19)}}\ [77]$	$\mathit{47_{(25)}}:378_{(19)}\ [\mathit{192}]$	$33_{(25)}:269_{(19)}\ [123]$	$35_{(23,25)}:\mathbf{180_{(20)}}\ [87]$	$\mathit{43_{(25)}}:\mathit{463_{(19)}}\ [\mathit{213}]$
FFDProd	$22_{(23)}:\mathbf{124_{(19)}}\ [66]$	$30_{(23)}:151_{(15,19)}\ [83]$	$25_{(23)}:\mathbf{140_{(19)}}\ [74]$	$28_{(24)}:206_{(20)}\ [86]$	$29_{(23)}:213_{(19)}\ [89]$
FFDSum	$\mathit{27_{(21)}}:128_{(19)}\ [73]$	$38_{(25)}:251_{(17,18)}\ [128]$	$32_{(25)}:165_{(17)}\ [91]$	$30_{(24)}:263_{(20)}\ [110]$	$33_{(25)}:264_{(19)}\ [116]$
DotProd	$25_{(21)}:127_{(19)}\ [70]$	$31_{(25)}:153_{(19)}\ [85]$	$28_{(23)}:141_{(19)}\ [76]$	$29_{(24)}:204_{(20)}\ [89]$	$30_{(25)}:220_{(19)}\ [96]$
RBP	$\mathbf{19_{(22)}}:\mathbf{124_{(19)}}\ [\mathbf{60}]$	$\mathbf{22_{(22)}}:\mathbf{150_{(19)}}\ [\mathbf{72}]$	$\mathbf{20_{(22)}}:\mathbf{140_{(19)}}\ [\mathbf{66}]$	$\mathbf{26_{(22)}}:207_{(20)}\ [\mathbf{79}]$	$\mathbf{24_{(22)}}:213_{(19)}\ [\mathbf{82}]$

	PM Set 6 min:max [average]	PM Set 7 min:max [average]	PM Set 8 min:max [average]	PM Set 9 min:max [average]	PM Set 10 min:max [average]
FFDCPU	$\mathit{21_{(24)}}:\mathit{134_{(20)}}\ [68]$	$\mathit{52_{(24)}}:\mathit{336_{(20)}}\ [\mathit{173}]$	$\mathit{35_{(24)}}:\mathit{203_{(20)}}\ [\mathit{107}]$	$\mathit{53_{(24)}}:\mathit{339_{(20)}}\ [\mathit{183}]$	$25_{(24)}:\mathbf{133_{(19)}}\ [64]$
FFDRAM	$\mathit{21_{(25)}}:133_{(19)}\ [\mathit{69}]$	$33_{(25)}:\mathbf{149_{(19)}}\ [82]$	$33_{(25)}:200_{(19)}\ [106]$	$25_{(22,23)}:\mathbf{133_{(20)}}\ [64]$	$\mathit{51_{(25)}}:\mathit{350_{(19)}}\ [\mathit{186}]$
FFDProd	$19_{(23)}:\mathbf{98_{(20)}}\ [54]$	$22_{(23)}:160_{(20)}\ [65]$	$20_{(23)}:\mathbf{101_{(20)}}\ [57]$	$21_{(23)}:147_{(20)}\ [65]$	$22_{(23)}:148_{(19)}\ [65]$
FFDSum	$\mathit{21_{(22,23)}}:99_{(19)}\ [58]$	$24_{(22,23)}:178_{(20)}\ [70]$	$23_{(21,22)}:112_{(17)}\ [63]$	$24_{(21,22,23)}:183_{(20)}\ [72]$	$25_{(25)}:187_{(19)}\ [75]$
DotProd	$\mathit{21_{(21)}}:\mathbf{98_{(19,20)}}\ [57]$	$24_{(21)}:158_{(20)}\ [65]$	$23_{(21)}:103_{(17)}\ [60]$	$23_{(21,23)}:148_{(20)}\ [65]$	$24_{(23)}:150_{(19)}\ [70]$
RBP	$\mathbf{17_{(22)}}:\mathbf{98_{(20)}}\ [\mathbf{50}]$	$\mathbf{20_{(22)}}:160_{(20)}[\mathbf{60}]$	$\mathbf{18_{(22)}}:\mathbf{101_{(20)}}\ [\mathbf{52}]$	$\mathbf{19_{(22)}}:148_{(20)}\ [\mathbf{60}]$	$\mathbf{19_{(22)}}:148_{(19)}\ [\mathbf{61}]$

	PM Set 11 min:max [average]	PM Set 12 min:max [average]	PM Set 13 min:max [average]	PM Set 14 min:max [average]	PM Set 15 min:max [average]
FFDCPU	$22_{(24)}:\mathit{135_{(20)}}\ [\mathit{68}]$	$\mathit{41_{(24)}}:\mathit{303_{(20)}}\ [148]$	$\mathit{31_{(24)}}:208_{(20)}\ [\mathit{108}]$	$\mathit{51_{(24)}}:\mathit{366_{(20)}}\ [\mathit{177}]$	$29_{(23,24)}:\mathbf{160_{(19)}}\ [75]$
FFDRAM	$22_{(25)}:\mathit{135_{(19)}}\ [\mathit{68}]$	$40_{(25)}:\mathit{303_{(19)}}\ [\mathit{151}]$	$\mathit{31_{(25)}}:\mathit{212_{(19)}}\ [100]$	$25_{(25)}:\mathbf{139_{(20)}}\ [68]$	$\mathit{44_{(25)}}:\mathit{388_{(19)}}[\mathit{179}]$
FFDProd	$20_{(23)}:\mathbf{103_{(20)}}\ [57]$	$24_{(23)}:127_{(16)}\ [68]$	$20_{(23)}:\mathbf{109_{(19)}}\ [60]$	$23_{(23)}:158_{(20)}\ [69]$	$25_{(23)}:183_{(19)}\ [75]$
FFDSum	$\mathit{23_{(21,23)}}:107_{(20)}\ [62]$	$29_{(25)}:203_{(18)}\ [95]$	$26_{(21,22,23,25)}:122_{(17)}\ [68]$	$27_{(24)}:211_{(20)}\ [85]$	$25_{(25)}:252_{(19)}\ [98]$
DotProd	$22_{(21)}:\mathbf{103_{(20)}}\ [60]$	$27_{(21)}:128_{(16)}\ [71]$	$23_{(21)}:110_{(19)}\ [62]$	$25_{(21,23)}:157_{(20)}\ [69]$	$26_{(25)}:190_{(19)}\ [82]$
RBP	$\mathbf{17_{(22)}}:\mathbf{103_{(20)}}\ [\mathbf{52}]$	$\mathbf{20_{(22)}}:\mathbf{118_{(20)}}\ [\mathbf{59}]$	$\mathbf{18_{(22)}}:\mathbf{109_{(19)}}\ [\mathbf{54}]$	$\mathbf{19_{(22)}}:159_{(20)}\ [\mathbf{62}]$	$\mathbf{20_{(22)}}:183_{(19)}\ [\mathbf{70}]$

	PM Set 16 min:max [average]	PM Set 17 min:max [average]	PM Set 18 min:max [average]	PM Set 19 min:max [average]	PM Set 20 min:max [average]
FFDCPU	$\mathit{23_{(24)}}\ :\mathit{149_{(20)}}\ [\mathit{75}]$	$\mathit{59_{(24)}}\ :327_{(20)}\ [176]$	$\mathit{41_{(24)}}\ :239_{(20)}\ [\mathit{126}]$	$\mathit{66_{(24)}}\ :\mathit{405_{(20)}}\ [\mathit{216}]$	$24_{(24)}\ :\mathbf{114_{(20)}}\ [65]$
FFDRAM	$21_{(25)}\ :142_{(19)}\ [71]$	$55_{(25)}\ :\mathit{330_{(19)}}\ [\mathit{187}]$	$39_{(25)}\ :\mathit{246_{(19)}}\ [\mathit{126}]$	$23_{(25)}\ :\mathbf{112_{(19)}}\ [62]$	$\mathit{67_{(25)}}\ :\mathit{403_{(19)}}\ [\mathit{213}]$
FFDProd	$19_{(23)}\ :\mathbf{97_{(19)}}\ [54]$	$21_{(23)}\ :\mathbf{106_{(20)}}\ [59]$	$19_{(23)}\ :\mathbf{100_{(19)}}\ [56]$	$20_{(23)}\ :116_{(20)}\ [58]$	$21_{(23)}\ :130_{(19)}\ [61]$
FFDSum	$21_{(23)}\ :\mathbf{97_{(19)}}\ [57]$	$23_{(22,23)}\ :123_{(17)}\ [67]$	$22_{(22)}\ :103_{(19)}\ [60]$	$23_{(21,22,23)}\ :125_{(20)}\ [64]$	$26_{(21,22)}\ :141_{(21)}\ [68]$
DotProd	$21_{(21)}\ :98_{(19)}\ [57]$	$23_{(21,23)}\ :109_{(20)}\ [61]$	$22_{(21)}\ :101_{(19)}\ [58]$	$23_{(21)}\ :116_{(20)}\ [60]$	$24_{(21)}\ :134_{(19)}\ [66]$
RBP	$\mathbf{17_{(22)}}\ :\mathbf{97_{(19)}}\ [\mathbf{50}]$	$\mathbf{18_{(22)}}\ :\mathbf{106_{(20)}}\ [\mathbf{53}]$	$\mathbf{17_{(22)}}\ :\mathbf{100_{(19)}}\ [\mathbf{51}]$	$\mathbf{18_{(22)}}\ :116_{(20)}\ [\mathbf{54}]$	$\mathbf{18_{(22)}}\ :130_{(19)}\ [\mathbf{57}]$

	PM Set 21 min:max [average]	PM Set 22 min:max [average]	PM Set 23 min:max [average]	PM Set 24 min:max [average]	PM Set 25 min:max [average]
FFDCPU	$29_{(24)}\ :\mathit{189_{(20)}}\ [91]$	$35_{(24)}\ :419_{(16)}\ [201]$	$36_{(24)}\ :285_{(20)}\ [131]$	$\mathit{47_{(24)}}\ :633_{(20)}\ [\mathit{244}]$	$39_{(22,25)}\ :630_{(19)}\ [156]$
FFDRAM	$30_{(23)}\ :186_{(19)}\ [89]$	$\mathit{51_{(25)}}\ :412_{(16)}\ [199]$	$32_{(25)}\ :\mathit{294_{(19)}}\ [128]$	$40_{(23)}\ :647_{(20)}\ [154]$	$\mathit{40_{(25)}}\ :615_{(19)}\ [\mathit{232}]$
FFDProd	$26_{(23)}\ :\mathbf{161_{(20)}}\ [81]$	$34_{(24)}\ :407_{(16)}\ [132]$	$32_{(23)}\ :200_{(16)}\ [96]$	$\mathbf{30_{(24)}}\ :580_{(20)}\ [162]$	$\mathbf{27_{(25)}}\ :553_{(19)}\ [159]$
FFDSum	$\mathit{31_{(21,23)}}\ :171_{(17,20)}\ [\mathit{93}]$	$47_{(24)}\ :\mathit{619_{(18)}}\ [\mathit{245}]$	$\mathit{38_{(24)}}\ :285_{(18)}\ [\mathit{139}]$	$40_{(24)}\ :\mathit{677_{(20)}}\ [223]$	$33_{(25)}\ :\mathit{654_{(19)}}\ [205]$
DotProd	$28_{(21,23)}\ :163_{(19)}\ [85]$	$35_{(24)}\ :415_{(16)}\ [143]$	$34_{(22,23,24)}\ :201_{(16)}\ [103]$	$\mathbf{30_{(24)}}\ :654_{(20)}\ [165]$	$29_{(25)}\ :562_{(19)}\ [172]$
RBP	$\mathbf{20_{(22)}}\ :162_{(20)}\ [\mathbf{70}]$	$\mathbf{26_{(22)}}\ :\mathbf{301_{(16)}}\ [\mathbf{102}]$	$\mathbf{22_{(22)}}\ :\mathbf{185_{(19)}}\ [\mathbf{81}]$	$\mathbf{30_{(24)}}\ :\mathbf{547_{(20)}}\ [\mathbf{146}]$	$\mathbf{27_{(25)}}\ :\mathbf{540_{(19)}}\ [\mathbf{150}]$

of each result. The results of l2norm, *RBP*$_c$ *(*$\alpha = 2$*)* and *RBP*$_c$ *(*$\alpha = 10$*)* are not presented because l2norm has the same results as Dot-Product (see Section V-B), and *RBP*$_c$ *(*$\alpha = 2$*)* and *RBP*$_c$ *(*$\alpha = 10$*)* have results very close to but do not outperform *RBP* as shown in Figure 6. The results show that a heuristic (e.g., FFDRAM) which performs the best in one dataset (e.g., (PM set 7, VM set 19)) can become the worst in another dataset (e.g., (PM set 5, VM set 19). *RBP* has the best minimum and average results through the 25 PM sets. 10 of *RBP's* maximum results are not the smallest. However, none of them is the worst. The biggest difference between *RBP's* maximum result and the best maximum result is 15.76%.

C. Comparison with CPLEX

To investigate how close *RBP*'s results are from optimal results, we compare it with CPLEX, one of the leading optimisation problem solvers.

Figure 7 shows that *RBP* generates results close to CPLEX with the biggest gap as 13.58% and average gap as 5.54%.

The execution time of *RBP* is less than one second when executed using the small IBM dataset [5]. The scalability of *RBP* is evaluated using the set of PMs from the big IBM

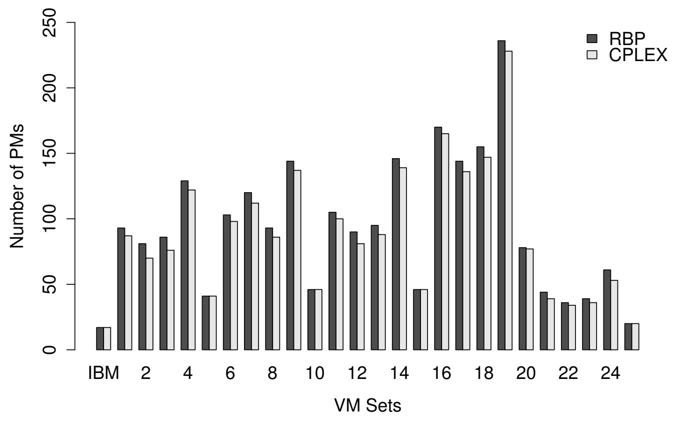


Fig. 7. Comparing Number of Hosts required by *RBP* and CPLEX against (i) the small IBM dataset and (ii) different VM sets (from the benchmark) and the PM set of the big IBM dataset.

dataset and duplicating the VM set of the small IBM dataset ($2,869$ VMs) in order to obtain various number of VMs. Figure 8 shows that *RBP* is orders of magnitude better than CPLEX when the size of the dataset scales up. When the size of VM set increases to $8,607$, CPLEX failed to produce a solution within acceptable time (i.e., more than 10 hours) while *RBP* requires 5.9 seconds to execute.

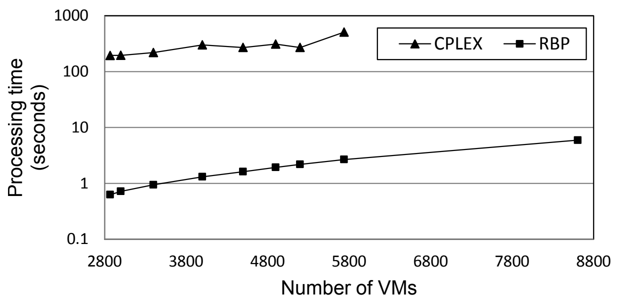


Fig. 8. Scale-up experiement: execution time of *RBP* and CPLEX (note the logarithmic scale) with 342 PMs and different number of VMs.

VI. Conclusion and Future Work

This paper proposes *RBP*, a VM placement heuristic and a benchmark for a fair comparison of VM placement algorithms. *RBP* is validated using both real industrial traces and synthetic data. It is compared with commonly used heuristics and optimal solutions, and is proved to be able to provide a close to optimal solution in a few seconds even when the dataset scales up. The evaluation experiments also suggest that the dataset demographics have an impact on the results of VM placement heuristics, and the results presented in this paper can provide a hint in selecting a heuristic according to the data center's infrastructure and workload structure.

For future work, it would be interesting to evaluate VM placement algorithms further using more datasets extended from our benchmark. Moreover, we intend to investigate the interference between co-located VMs at a finer granularity to achieve an even more efficient utilization when alleviating performance degradation.

Acknowledgment

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie) and by Enterprise Ireland Innovation Partnership in cooperation with IBM and University College Dublin under grant IP/2010/0061.

References

[1] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," in *Proc. ASPLOS XIV*, 2009, pp. 205–216.

[2] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, Jan. 2008.

[3] T. Cioara, I. Anghel, I. Salomie, G. Copil, D. Moldovan, and A. Kipp, "Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning," in *Proc. ISPDC*, 2011, pp. 163–169.

[4] J. Xu and J. A. B. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *Proc. GREENCOM-CPSCOM*, 2010, pp. 179–188.

[5] X. Li, A. Ventresque, N. Stokes, J. Thorburn, and J. Murphy, "ivmp: An interactive vm placement algorithm for agile capital allocation," in *Proc. CLOUD*, 2013, pp. 950–951.

[6] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating Heuristics for Virtual Machines Consolidation," Microsoft Research, Tech. Rep., 2011.

[7] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proc. HotPower*, 2008, pp. 10–10.

[8] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. NSDI*, 2007, pp. 17–17.

[9] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: Insights from google compute clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.

[10] "Ibm ilog cplex optimizer," accessed: 2014-03-16. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html

[11] "Best practices for identifying and resolving infrastructure problems," accessed: 2014-03-16. [Online]. Available: http://www-03.ibm.com/software/products/us/en/tivomoni/

[12] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective vm sizing in virtualized data centers," in *Proc. IFIP/IEEE IM*, 2011, pp. 594–601.

[13] C. Isci, J. Hanson, I. Whalley, M. Steinder, and J. O.Kephart, "Runtime demand estimation for effective dynamic resource management," in *Proc. NOMS*, 2010, pp. 381–388.

[14] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proc. INFOCOM*, 2014.

[15] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Proc. CloudCom*, 2009, pp. 254–265.

[16] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. USENIX ATC*, 2009, pp. 28–28.

[17] E. Feller, L. Rilling, and C. Morin, "Energy-aware ant colony based workload placement in clouds," in *Proc. GRID*, 2011, pp. 26–33.

[18] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters," in *Proc. ACM SIGPLAN/SIGOPS VEE*, 2009, pp. 41–50.

[19] K. Tsakalozos, M. Roussopoulos, and A. Delis, "Vm placement in non-homogeneous iaas-clouds," in *Proc. ICSOC*, 2011, pp. 172–187.

[20] S. K. Bose and S. Sundarrajan, "Optimizing migration of virtual machines across data-centers," in *Proc. ICPPW*, 2009, pp. 306–313.

[21] S. Steven, *The Algorithm Design Manual*. Springer, 2010.

[22] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109–1130, 2007.