PeerJ Computer Science

Decomposed multi-objective bin-packing for virtual machine consolidation

Eli M. Dow

Industries and Solutions, IBM Research, Yorktown Heights, NY, United States
Department of Interdisciplinary Engineering Science, Clarkson University, Potsdam, NY, United States

ABSTRACT

In this paper, we describe a novel solution to the problem of virtual machine (VM) consolidation, otherwise known as VM-Packing, as applicable to Infrastructure-as-a-Service cloud data centers. Our solution relies on the observation that virtual machines are not infinitely variable in resource consumption. Generally, cloud compute providers offer them in fixed resource allocations. Effectively this makes all VMs of that allocation type (or instance type) generally interchangeable for the purposes of consolidation from a cloud compute provider viewpoint. The main contribution of this work is to demonstrate the advantages to our approach of deconstructing the VM consolidation problem into a two-step process of multidimensional bin packing. The first step is to determine the optimal, but abstract, solution composed of finite groups of equivalent VMs that should reside on each host. The second step selects concrete VMs from the managed compute pool to satisfy the optimal abstract solution while enforcing anti-colocation and preferential colocation of the virtual machines through VM contracts. We demonstrate our high-performance, deterministic packing solution generation, with over 7,500 VMs packed in under 2 min. We demonstrating comparable runtimes to other VM management solutions published in the literature allowing for favorable extrapolations of the prior work in the field in order to deal with larger VM management problem sizes our solution scales to.

Subjects Autonomous Systems, Operating Systems
Keywords Virtual machine management, Consolidation, IaaS

Submitted 9 July 2015
Accepted 3 February 2016
Published 24 February 2016

Corresponding author
Eli M. Dow, emdow@us.ibm.com

Academic editor
Wolfgang Banzhaf

Additional Information and Declarations can be found on page 22

DOI **10.7717/peerj-cs.47**

INTRODUCTION AND MOTIVATION

The primary contributions of this work are the construction of an efficient algorithm for packing virtual machines densely in an Infrastructure-as-a-Service (IaaS) data center. Reducing the number of physical hosts is one way to reduce costs of cloud computing data centers by limiting the power and cooling needs of servers that need not run at full capacity when workloads can be run on a subset of the total available hardware. As has been demonstrated in the literature, powering down hosts completely is generally not ideal given the dynamic nature of cloud computing workloads. Consider that some number of individual cloud servers may participate in distributed services that necessarily preclude turning the machines off entirely due to response requirements when serving file contents. Though shutting hosts off completely is not an option in all cases, idle machines can be placed into a low power state to achieve much of the benefits described earlier (*Isci et al., 2011*; *Zhang et al., 2014*). Similar concepts have been applied to the persistent

disk volumes supporting storage networks for clouds (*Narayanan, Donnelly & Rowstron, 2008*). This effort is also supplementary to an ongoing cloud management framework being developed by our research group specifically to suit small-to-medium business-sized IaaS compute clouds.

Average data center server utilization is reportedly estimated to be around 10% (±5%) (*United States Environmental Protection Agency, 2007*). Research shows that such low utilization is inefficient because even idle servers may consume more than 50% of their peak power requirements (*Gandhi et al., 2009*). The result of these findings is the implication that data centers should strive to have fewer servers running at close to peak utilization if they want to be efficient and more competitive on cost pricing. An obvious solution to drive down cloud compute cost is therefore to consolidate cloud compute resources (Virtual machines) onto the minimum number of host servers.

Many of the previous efforts to describe solutions to Infrastructure-as-a-Service (IaaS) virtual machine packing stem from previous analyses and solutions to general purpose bin-packing type problems. Conceptually it works by assuming each VM is allocated some fixed share of the physical host's compute resources. Examples of those fixed resources include the number of logical cores on a multi-core host, or the fractional allocation of the host's total available memory and persistent storage. Many of these approaches are variations on the theme of multi-dimensional bin-packing approach (*Campegiani & Lo Presti, 2009*; *Panigrahy et al., 2011*). Others are based on genetic algorithms such as *Gu et al. (2012)*, *Campegiani (2009)*, and still others on market based systems (*Lynar, Herbert & Simon, 2009*).

Our goal is to design a consolidation placement algorithm that is high performance and suitable for small-to-medium business-sized virtual machine data centers on the order of a few hundred virtual machines spanning fewer than 100 hosts. We focus on this demographic for two reasons. The first is because of the tractability of the problem. The second is because the medium sized business users are less likely to opt for exorbitant hardware capital expense to maintain performance while attempting some level of consolidation. Measurements on Google, Amazon, and Microsoft cloud service platforms demonstrate revenue losses ranging from 1 to 20% (*Greenberg et al., 2009*; *Kohavi, Henne & Sommerfield, 2007*; *Schurman & Brutlag, 2009*) due to consolidation induced latency. As such, the super large data centers often under consolidate intentionally to maintain response times. In Google data centers, for instance, it has been reported that consolidated workloads make use of only half of the available compute cores on the hardware (*Mars et al., 2011*). Every other processor core is left unused simply to ensure that performance does not degrade. Small-to-medium business (SMB) operations on the other hand, may be less cost sensitive to minute latency introductions, especially if they are supporting internal private cloud data centers on premises.

PRIOR SOLUTIONS FOR GENERAL BIN PACKING

When considering the existing solutions to virtual machine consolidation placement, it is reasonable to conclude that the problem is most similar to bin-packing solutions.

Stated in terms of the classical Bin Packing problem, virtual machine consolidation is stated as such: given a set $I = \{1, \ldots, n\}$ of items, where item $i \in I$ has size $s_i \in (0, 1]$ and a set $B = \{1, \ldots, n\}$ of bins with capacity one. Find an assignment a: $I \rightarrow B$ such that the number of empty bins is maximized. The problem is known to be NP-complete. Further it has been shown that there is no ρ-approximation algorithm with $\rho < 3/2$ for Bin Packing unless $P = NP$.

In other words, consider a finite collection of n weights called $w(w_1, w_2, w_3, \ldots, w_n)$, and a collection of identical bins with capacity C (that necessarily must be larger than the maximum individual weight from the set w), what is the minimum number (k) of bins required to contain all the weights without exceeding the bin capacity C? Setting aside the mathematical formulation, we are interested in the fewest bins required to store a collection of items. This variant of the bin-packing problem is known as the 1-dimensional (1-D) bin packing problem. Note that it is important to keep in mind that "weights" here are necessarily formulated as indivisible atomic values. The weight of an object being placed must fit entirely within the remaining capacity of a container and may not be subdivided.

Approximation solutions to the bin packing problem usually make use of heuristic approaches, with the two most relevant solutions being *Next-Fit*, and *First-Fit Decreasing*.

Next-Fit

In the *Next-Fit* algorithm, all bins are initially considered to be empty. Beginning with bin $j = 1$ and item $i = 1$, if bin j has residual capacity for item i, assign item i to bin j via an assignment function a, i.e., $a(i) = j$. After an assignment is made, simply consider the next item $(i+1)$ until exhausting all items. If there are items that still require packing, consider bin $j+1$ and item i. This process continues until the last item is assigned (packed).

The major drawback of the *Next-Fit* algorithm is that the approach does not consider a bin after it has been processed once and left behind. This can lead to wasted capacity in those bins that are not revisited, and thus there is room for finding a better approximation solution.

First-Fit Decreasing

One solution to the drawbacks inherent in *Next-Fit*, is to employ the following algorithm. At the start, consider each of k bins as empty. Beginning with bins $k = 1$ and item $i = 1$, consider all bins $j = 1, \ldots, k$ and place item i in the first bin that has sufficient residual capacity (i.e., $a(i) = j$). If there is no such bin available, increment k and repeat until the last item n is assigned.

This form of *First-Fit* provably uses at most $17/10 * k$ bins, where k is the optimal number of required bins for a perfect solution (*Johnson et al., 1974*). However there are further refinements to the algorithm that should be considered.

The simple intuition behind *First-Fit Decreasing* is that if you first reorder the items such that $s1 \geq \cdots \geq sn$ before applying the general *First-Fit* algorithm, you can obtain a superior result. Intuitively this is because the conceptually "larger" items, in our case multiple large resource consumption VMs are less likely to fit onto a single bin (host),

so we place each of the large resource consumers first, using the residual space to pack in smaller virtual machines. First fit decreasing runs in $O(n^2)$ time.

Multi-dimensional Bin-Packing

A key concept in 1-D Bin-Packing is that there is a single unified weight to be used as the driving input to packing constraints. But in the real world, bin packing is often subject to multivariate constraints. Consider trucks being packed with boxes for shipment. We could use a 1D bin-packing approach based on box weight and the shipping weight of a fleet of trucks, but this doesn't work out well in the real world because boxes have a 3-dimensional volume to consider in addition to a weight. Simply consider many small but very heavy boxes, or a few light boxes that are voluminous and things become more complicated.

HIERARCHICAL MULTI-OBJECTIVE BIN PACKING FOR VIRTUAL MACHINES

Our solution to packing virtual machines is predicated on two major observations. The first is that most virtual machine cloud providers offer fixed size virtual machines at varying cost structures. At the time of this writing Amazon, one of the largest cloud computing hosts for IaaS virtual machines, offers VM sizes in small, medium, large, extra-large, double-extra-large, and quadruple-extra-large capacities, each with their own pricing according to a tiered cost structure. This means that in some sense virtual machines belonging to the same capacity and cost structure tier can be considered equivalent for the purposes of packing assignment.

The intuition here is that virtual machine packing in a cloud data center is definitely a multi-dimensional bin-packing type of problem where the constraints are related to the host capacities and virtual machine consumption requirements of memory, CPU, disk I/O, and networking.

Our formulation of the bin packing problem, as it applies to virtual machines, is as follows. Given a host with overall hardware capacity C, we can represent that capacity as a tuple of the individual resources that comprise C, for instance CPU, RAM, and IO as constituent individual capacities, c_i we can say formalize the capacity of a host as:

$$C = \{c_1, c_2, c_3, \ldots, c_n\} \quad \text{s.t. } \forall c_i \in C, c_i \geq 0$$

where the physical capacities of a limited resource, i.e., c_1, might representing the physical CPU capacity of the host, c_2 representing RAM capacity of the host, and c_3 representing I/O capacity of host with capacity tuple C.

A virtual machine, V can similarly be defined as a tuple of resources required for the operation of that VM according to the same resource cardinality used to describe resource capacities of a host. As an example here:

$$V = \{v_1, v_2, v_3, \ldots, v_n\}.$$

The value v_1 represents the CPU requirement of virtual machine V, while v_2 similarly means the RAM requirements of the VM, and v_3 means the I/O requirements of the virtual machine V.

With this formalism we can further consider that a given host is considered over-constrained if any per-resource cumulative VM utilization exceeds a maximum percentage τ of the physical resource a given threshold. If no oversubscription of resources is allowed with the virtualization platform, $(0 < \tau \leq 1)$, if oversubscription of resources is permissible, $(0 < \tau \leq \tau_{\text{MAX}})$ where τ_{MAX} is the maximum oversubscription threshold allowed.

There is thus a tuple for defining the per-host (C), per-measured resource utilization/over-constraint percentage (τ_i) of the form:

$$\tau_c = \{\tau_1, \tau_2, \tau_3, \ldots, \tau_n\}.$$

Accordingly, each measured resource may have a τ_{MAX} defined according to the same cardinality as C and V that denotes the maximum per-resource utilization/over-constraint percentage for a given measured resource. We thusly define functions to retrieve the over-constraint factor as follows: $\tau(C, i)$ which is alternatively written as $\tau_c[i]$ using array indexing notation, along with and $\tau_{\text{MAX}}(C, i)$ which is alternatively written as $\tau_{\text{MAXc}}[i]$, where C remains a specific host instance, and i remains the cardinality of some measured resource.

If there are k virtual machines assigned or allocated to a given host C, the boolean packing allocation $P(C)$ is defined as follows:

$$P(C) = \begin{cases} 1, & \text{iff } \sum_{i=1}^{k} v_i \leq ci * \tau(c, i) \leq ci * \tau_{\text{MAX}}(c, i) \\ 0, & \text{iff } \sum_{i=1}^{k} v_i > ci * \tau(c, i). \end{cases}$$

The general bin packing problem attempts to solve for the minimum number of hosts n, s.t. $\forall V_i \in V$, V_i is placed on some host $C_j \in C$. Hosts which contain some number of packed VMs form the set C' with $C' \subseteq C$. Any elements in $C \notin C'$ are unassigned or unused and may be powered down or put into low power standby. Thus, the goal of bin packing is to determine the smallest n number of hosts with one or more VMs allocated, and the allocations are valid according to our packing validity function (P) above.

$$\text{Min}(n) \text{ s.t.} \begin{cases} n = |C'| \\ \forall Cn \in C, V(C) = 1. \end{cases}$$

The problem with the general bin packing problem is that it is computationally prohibitive to compute all possible packing solutions to perform this minimization as the combinations of VM to host mappings explodes. In contrast to previous virtual machine bin packing solutions, our approach first generates a generic, yet optimal solution to a slightly different problem that relaxes the condition that each VM be considered

independently or seen as unique in its resource consumption requirements. Specifically it determines, for each host, the required allocation count from a finite (read arbitrarily small) enumeration of similar sized VMs (VM equivalence sets). This arbitrarily small grouping of similar sized VMs is based on partitioning, or clustering VMs into equivalence sets according to resource consumption size. By relaxing the uniqueness of VMs and allowing groups of them to be treated identically in terms of assumed, nominal, resource consumption, an optimal solution to this problem can be found very rapidly using a dynamic programming approach (described in detail in 'Constructing an ideal solution template'). Once this generic recipe for VM allocation is determined, it is concretely fulfilled from the actual list of distinct virtual machines being managed by employing a variant of *First-Fit Decreasing* (as described in 'Concrete VM allocation fulfillment). The first-fit decreasing approach selects concrete VM instances from the appropriate size peer-groups using a hierarchical multi-objective strategy, which is tunable, by the system operator depending on requirements. This latter multi-objective fulfillment is and integral part of our solution because we respect the notion that bad colocation selections can impair overall system performance. Consider for instance the prior research on process colocation scheduling that uses previously measured performance degradations of possible combinations as input to colocation placement decisions to yield the lowest possible co-located degradation (*Jiang et al., 2008*; *Jiang, Tian & Shen, 2010*; *Tian, Jiang & Shen, 2009*). This work was improved upon and made more relevant to the class of virtual machine consolidation problems by an approach that enabled an explicit, tunable performance bound in PACMan (*Roytman et al., 2013*) as a goal for consolidation. However, the work in PACMan required initial placement to be on lightly consolidated servers (conservative placement in their terminology), for a period of time while observations were made to determine colocation impact characteristics on the VM. The authors of the PACMan system used 30–60 min as example batch periods for this purpose. In contrast, our goal was to build something that can perform VM consolidation packing almost continuously (every few minutes), with low resource usage.

Constructing an ideal solution template

To construct a template for what the ideal packing solution should look like, our algorithm uses a dynamic programming style solution. The first step is to discretize the virtual machines to be packed into a set of equivalence combinations. Equivalence here means more or less the same computational requirements. For instance, if one is familiar with the Amazon EC2 Instance types, one could consider all instantiations of an EC2 Instance type to be equivalent for the purposes of virtual machine bin packing. In some situations, the size of equivalence sets combinations are known *a priori* because virtual machines are cloned from templates, or golden master images and then customized for each unique deployment. In general, one can consider our algorithm as "off-line" in bin-packing parlance because we know the sizes of all containers and all objects to be packed prior to running the algorithm.

The equivalence set can be represented by an ordered tuple consisting of the largest, per-VM resource consumption parameters from each element in the set. For instance,

in our experimentation we used the following notation to describe an equivalence set: "M::16 C::8 N::2 S::1 E::2" which describes the set of instances containing 16 GB RAM, eight logical CPU cores, two network compute units, a single set holding of two concrete elements (VMs) in total. Later versions of this software were enhanced to more specifically separate out the network values into a cumulative total, as well as constituent read and write capacity. The same approach was also implemented for disk I/O, but has been omitted from the examples for brevity.

In situations where the equivalence sizes are not known ahead of time, one can opt to perform a clustering approach to define equivalence groupings, and thereby limit the groupings to a suitably small number of equivalence combinations so as to make this approach viable. Once a cluster is defined using a technique such as K-means clustering one can define the equivalence set to be each VM in the grouping.

Likewise, the hosts in the cloud data center can be partitioned into equivalence sets. Hosts belong to the same equivalence set when any two hosts in the set are considered logically equivalent to each other in terms of capacity. When large cloud providers procure machines, they are often purchased several at a time according to the same specification, thereby lending itself to this approach well. In situations where hosts are not identical machines, system administrators can broadly lump together machines with similar compute capabilities.

The algorithm begins by looking at each of the virtual machine equivalence combinations and examining the representative capacity. The algorithm then iteratively generates *starting sets* representing all possible combinations consisting of only virtual machines from the same equivalence (from the set of similarly sized VMs) that are viable (would fit logically on the largest host to be filled).

The limiting factor that governs the expansion of the starting sets is when the resource requirements grow too large to fit on the largest capacity host. Consider the following example of this is unclear: assume the largest capacity host has 1 GB of RAM, and the smallest element from an equivalence set of virtual machines, lets call that set the SMALL instances, each require 256 MB of RAM. In this case, the starting sets produced from the SMALL instances would be the following trivial generation: 1×SMALL, 2×SMALL, 3×SMALL, and 4×SMALL as each set would cumulatively require 256, 512, 768, and 1,024 MB of RAM, respectively. No larger grouping of SMALL instances would fit on the largest host. These starting sets represent some initial candidate equivalence combinations to consider.

Naturally, we extend this to include an oversubscription parameter for each resource dimension (RAM, CPU, disk I/O, networking I/O, etc.) that allows the resource to be oversubscribed on a per-host basis. From the example above, considering no oversubscription for RAM, we would have generated all four of the possible instantiations enumerated. Each element in the starting set represents an abstract combination from the SMALL-VM set that could be satisfied by concrete virtual machines that are considered equivalent.

We further limit the generation of abstract combinations to ensure that combinations respect each dimension of resource under consideration. For instance, in the running

example there might actually only be three abstract combinations generated due to the cumulative I/O or CPU requirements of the abstract grouping as compared to the host capacity. We also limit the abstract combinations to contain no more of an equivalence set than the concrete instances. In other words, we do not generate something like 4×SMALL if there are in fact only 3 instances of the small VM size to be packed.

The algorithm proceeds through each VM equivalence set (e.g., MICRO, SMALL, MEDIUM, LG, XL, XXL, QXL) generating all possible combinations of abstract VMs that could reside on the largest host on the network.

This process of creating starting sets, when used with a relatively small number of equivalence sets (the order of 10 or so), completes virtually instantaneously on modern machines. The number of *starting sets* is on the order of the number of virtual machine categories (equivalence sets provided as input).

With the completion of the generation of all abstract starting sets, we rigorously generate the exhaustive permutations of abstract, but viable, packing solutions. Expansion proceeds as follows. Assume that labels **a**, **b**, **c**, and **d** are labels for each of the generated equivalence sets. At step 1 the table of combinations could be listed as follows with a row dedicated to each known set where it must be strictly followed that a < b < c < d where the < operator is a standardized comparator, in our case RAM consumption:

a
b
c
d

At the next iteration, each row is combined with all elements below it:

a, ab, ac, ad
b, bc, bd
c, cd
d

Continuing in this fashion on a second iteration:

a, ab, ac, ad, abc, abd, acd
b, bc, bd, bcd
c, cd
d

And the final iteration yields the complete permutation list:

a, ab, ac, ad, abc, abd, acd, abcd
b, bc, bd, bcd
c, cd
d

As expected, we get 15 values from the original 4. This is because we are computing the combinations of $P(4,4) + P(4,3) + P(4,2) + P(4,1)$ following the standard $P(x,y)$

meaning "permutations of x values by choosing y elements." This is because, strictly speaking, virtual machine packing is a permutation problem not a combination problem since the meaning of a generic assignment solution that says: "2 large VMs and 1 small VM" is semantically identical to "1 small VM and 2 large VMs." Order does not matter, and repeats are not possible.

In reality we do not have single elements on each starting row however we have the starting sets expanded from the singleton abstract VMs. Each of the starting sets' elements are combined with every element not from its own starting set modulo invalid permutations. Invalid permutations limit the state expansion due to the limitation that candidate permutations must remain dimensionally smaller than the largest host (note that smaller here means that the candidate solution permutation must be constrained by each compute resource dimension being observed). We compute the exhaustive solution of abstract permutations using a dynamic programming approach.

By sorting the list of candidate solution recipes hierarchically as they are generated (for instance by cumulative memory requirement, then by cumulative CPU capacity required, etc.) we can eliminate much of the needless state expansion in computing all possible combinations through *short circuit evaluation*. The general worst-case runtime of computing bin-packing solutions, via an exhaustive computation of VM to host mapping possibilities is shown in Eq. (1):

$$\mathrm{O}\left(n^2\right) \tag{1}$$

where n is the number of VMs and k is the number of host servers. Instead, we compute something with complexity:

$$\mathrm{O}\left(\sum_{i=0}^{n} \frac{n!}{(n-i)!}\right). \tag{2}$$

In Eq. (2), we can see runtime of permutation combination based on virtual machine placement equivalence. In Eq. (2), n is the size of the set of equivalence combinations (in other words the number of VM instance types in Amazon parlance), and i is simply the counter from 1 to n. Alternatively, we could consider that this approach runs in time:

$$\mathrm{O}\left(n^k\right). \tag{3}$$

Equation (3) is an example of a second formulation of the runtime of permutation computation based on virtual machine placement equivalence. In Eq. (3), the value of n here is once more the size of the set of equivalence permutations (i.e., the number of VM instance types from Amazon). Since we know that the n here is strictly speaking a much smaller value than the n from Eq. (1) the base improvement shows a potential for speedup assuming k is greater than or equal to 2. Considering that the exponent term in Eq. (3) is a constant value of 2 rather than the size of the number of hosts as compared to the exponent in Eq. (1), this comparison of exponential terms suggests a massive speedup in computation time for the abstract solution as compared to the exhaustive concrete

solution for all situations where the data center has more than two hosts. The resulting implication is that the ability to compute the general form of a VM packing solution, if not the exact solution in light of equivalence partitions among the virtual machines is a reasonable strategy to pursue.

Note that during our experimentation we sorted the generated candidate solutions primarily on memory as it is shown in the literature to be the principal limiting factor in virtual machine consolidation (*Tulloch, 2010*; *VMware, Inc, 2006*).

The approach of generating the candidate recipe of what the ideal VM to host mapping solution should look like in the abstract is substantially faster than computing all possible concrete combinations of virtual machine assignments. Note that, at this point, when abstract solution generation is complete we are left with only a partial solution to the VM consolidation problem that informs us of what the ideal consolidation should look like in the abstract (e.g., 2×SMALL, 3×MEDIUM, 1×XL). However, as we will see in the next section, the choice to defer the actual runtime solution of concrete VM assignment to a second processing phase is advantageous.

Concrete VM allocation fulfillment

After phase one is complete and all candidate equivalence combinations have been generated, we now proceed to the concrete assignment phase. In this phase we begin by iterating through the list of hardware hosts. The list of hardware hosts is sorted in largest to smallest capacity.

We iterate through the sorted list of largest to smallest equivalence set permutations, assigning the first permutation that is satisfiable by the remaining, unassigned concrete VM instances. In essence this makes our algorithm a derivative of *First-Fit Decreasing* since we always fill the "largest" hosts first, with the "largest" equivalent set it can hold.

Once an equivalence set is found that fits the host being filled, the satisfier tries to locate concrete VM instances according to that equivalence set specification. If the set is not satisfiable (i.e., the equivalence set selected indicated 3 large VMs and there are only 2 remaining) we simply remove the candidate set from the list, and advance to the next "largest" candidate equivalence set. Note that largest here means the biggest according to our multi-objective comparator function used to sort the lists prior to satisfaction time.

In the assignment phase of consolidation, we have a roadmap for what the ideal consolidation should look like, but we still need to assign concrete virtual machine instances to each host that are representative of the plan. To do this, in theory, we could simply select any VM from the equivalence set specification (i.e., any SMALL VM is as good as any other). However, in practice we opt to use this opportunity to fulfill some hard and soft colocation/anti-colocation preferences expressed in VM contracts.

Listing 1: Example virtual machine definition with contract extensions. The *colocation_constraints* block demonstrates a single blacklisted host along with VMs for preferential colocation if possible.

```
<contract id="ubuntuvm">
 <ruleset type="vmm-libvirt">
  <domain type="kvm">
   <name>ubuntu</name>
   <uuid>03c03e0f-0a94-c167-5257-f52d772b1ec1</uuid>
   <memory>131072</memory>
   <currentMemory>131072</currentMemory>
   <vcpu>1</vcpu>
   <os>
      <type arch='x86_64' machine='pc'>hvm</type>
      <boot dev='hd'/>
   </os>
   <features>
      <acpi/>
   </features>
   <clock offset='utc'/>
   <on_poweroff>destroy</on_poweroff>
   <on_reboot>restart</on_reboot>
   <on_crash>destroy</on_crash>
   <devices>
     <emulator>/usr/bin/kvm</emulator>
     <disk type='file' device='disk'>
      <source file=
      '/home/deshantm/ubuntu-kvm/disk0.qcow2'/>
      <target dev='hda' bus='ide'/>
     </disk>
     <interface type='network'>
      <mac address='52:54:00:38:84:a7'/>
      <source network='default'/>
      <model type='virtio'/>
     </interface>
     <input type='mouse' bus='ps2'/>
     <graphics
        type='vnc'
        port='-1'
        autoport='yes'
        listen='127.0.0.1'/>
   </devices>
  </domain>
 </ruleset>
<vmclassifier>
     <workload type='cpu'/>
</vmclassifier>
<colocation_constraints>
     <blacklisted_colocation hosts=
   'f81d4fae-7dec-11d0-a765-00a0c91e6b12'/>
     <preferential_colocation vms=
           '448b138e-1ac6-4d45-ae9d-b5e735ac465f,
    a21e27e4-a8a8-4576-b770-8f8d0b893e71'/>
</colocation_constraints>
```

For instance, when selecting virtual machines we use a per-VM index of preferential colocation candidates and a per-VM set of anti-colocation constraints at satisfaction time. Anti-colocation are resolved first as they are considered a hard requirement, while colocation, or affinity preferences, are considered optional and satisfied when possible given the other firm constraints. Another aspect of our methodology is that we can then attempt more sophisticated placement systems such as load balancing across the consolidated hypervisors by using the actual runtime resource consumption of each virtual machine rather than just the nominal resource consumption maximums attributed to the equivalence group to which they belong. Listing 1 shows an example of a colocation constraint embedded in a virtual machine definition file for KVM as used in this research.

An overall review of our two-phased decomposition of the virtual machine bin packing problem can be seen in Algorithm 1.

Algorithm 1: Two-phase decomposition of the VM bin-packing wherein physical compute resource constraints are resolved in an abstract solution phase and colocation constraints are resolved during the second phase of concrete VM assignment.

Consolidate (MAXHOST, {MD}, {V}, {H})

1 Enumerate Equivalence Sets {E} from the set of all VMS {V} by method {E} = $E(\{V\})$

2 Expand singletons into singleton permutations:

3 **foreach** $i = 0$ **to** |{E}| **do**

4 **foreach** abstract combination element $e \in E_i$ **do**

5 {SE} = singletonExpansion(e)

6 append (E_i,{SE})

7 Compute all permutations {P} over every element in {SE}:

8 **for** $i = 0$ **to** |{SE}| **do**

9 P(|{SE}|, i)

Considering all measured dimensions $d, d \in \{MD\}$, discarding each permutation $p \in \{P\}$ where size (p_d) > capacity of the largest host (MAXHOST) on dimension d (MAXHOST(d)).

10 Apply First-Fit Decreasing algorithm with the set of hosts {H} as bins, and {P} as the elements to pack to determine a candidate allocation of abstract, equivalent VMs to a host:

11 **if** an element $p \in \{P\}$ is satisfiable by a single solution s **then** apply s.

12 **else if** a set of elements {C} satisfy $p \in \{P\}$, **then** apply greedy heuristics for anti-colocation and colocation to select a concrete set of virtual machines $c \in \{C\}$

NOTE: The sub-steps in 4 can be performed in such a way as to not generate invalid permutations and thereby reduce resource consumption at runtime.

RESULTS

To implement our algorithm, we began by outlining a small but efficient set of structures to encapsulate the state of a virtual machine based cloud datacenter. We implemented our software in the C programming language, using structs for objects representing the hardware machines and their physical limitations, as well as structs to represent the abstract virtual machines (something with a multidimensional size but no concrete identifier or

placement constraints attached). We also used a structure to represent equivalence sets, which were named lists of abstract VMs with aggregate resource requirements. In total, the source code was just under 2,000 lines of commented C code as determined by the Linux *wc* utility.

Once the VM and host information has been loaded from the preference file that describes the cloud to be modeled, the algorithm proceeds by enumerating all of the viable permutations of a single instance type. For instance, with the MICRO sized equivalence group modeled after Amazon's micro instance our enumeration of single-instance permutation would likely begin with the output as shown in Listing 2.

Listing 2: Example singleton permutation construction.

```
[(MICRO:1x1)-M::1 C::2 N::1 S::1 E::1]
[(MICRO:1x2)-M::2 C::4 N::2 S::1 E::2]
[(MICRO:1x3)-M::3 C::6 N::3 S::1 E::3]
      [...truncated for brevity...]
[(MICRO:1x48)-M::48 C::96 N::48 S::1 E::48]
[(SMALL:2x1)-M::2 C::1 N::1 S::1 E::1]
[(SMALL:2x2)-M::4 C::2 N::2 S::1 E::2]
      [...truncated for brevity...]
[(QXL:64x2)-M::128 C::52 N::2 S::1 E::2]
```

The format is simply saying the singleton type is micro, the 1×N notation indicates the number of instances, and the M, C, N, S, and E values represent the size of each of the memory, CPU, network, the number of sets in this combination (always 1 in a singleton set), and the number of concrete VM elements making up the set. Note, for simplicity of explanation, the dimension weights have been simplified to small, integer values. The listing might further continue for some time depending on the maximum capacity of the largest host or the maximum number of MICRO VM instances before continuing on to the SMALL sized VMs.

As an example of performance of our two-phased approach to solution generation for VM bin-packing, we performed a variety of experiments using a custom written virtual machine-based data center simulator. Our simulator is a Linux-based C program written to model various servers of varying physical capacities along the resource dimensions studied herein (NIC, RAM, CPU, etc...) along with virtual machines of varying sizes with per dimension resource over-constraint parameters and oversubscription detection/prevention on a per-resource basis. All simulator development and experimental validation was performed on IBM® Blade Center blades, each with 24-core CPUs and 94 GB RAM and later runs were repeated, on a 2014 Macbook Pro with 16 GB RAM. A simple XML configuration file specifying the virtual machines and hosts to be managed during the simulation governs the simulator. An example this configuration file is shown in the Appendix. The configuration file inputs include the per-resource over-constraint value. Overcommit is the oversubscription or over-allocation of resources to virtual machines

Table 1 Distribution of VMs across seven equivalence sets from a representative moderate-scale experiment.

VM equivalence set name	Concrete VM count
Micro	30
Small	21
Medium	10
Large	10
XL	5
DXL	3
QXL	1

with respect to the physical hosts maximum allocation of a particular resource. As an example, setting MEMORY overcommit to 1 means a 1:1 ratio of allowed cumulative VM memory consumption to the hosts physical RAM. Similarly, setting a CPU overcommit value of 2 indicates scheduling up to two virtual CPU cores. Decimal or floating-point values greater than one are accepted as valid inputs. In addition the configuration file for the simulation requires information about the largest host capacity for each measured resource, used for short circuit analysis when constructing abstract equivalence sets. In other words, the simulator enforces that no abstract solution is generated that has a cumulative individual resource consumption that exceeds the largest available host resource availability, for that resource (considering the previously specified over-constraint values for that resource). Lastly, the configuration file shown in Appendix includes information about the number of concurrent inbound and concurrent outbound migrations allowed as well as a cumulative migration maximum threshold per host. These migration threshold values are used for final solution post-processing that determines an automated live-migration based rearrangement plan to achieve a packed solution from an arbitrary starting state (VM to host allocation). The approach uses our previously described approach based on the well studied A∗ search algorithm to determine an automated migration plan (*Dow & Matthews, 2015*).

In a sample experiment that consisted of expanding a grouping of 79 concrete virtual machines as allocated according to Table 1, we generated 4,375,850 valid assignment permutations during the expansion phase (converting starting sets into all sets) in just 8 s of wall clock time. A further 14 s of wall clock time was used to perform final result sorting.

A larger scale experimental parameterization is shown in Table 2. During this experiment we performed a simulation of a large-scale cloud data center with 7,850 VMs. In this larger scale experiment, there were 174 starting sets generated, to ultimately create 13,465,212 permutations, taking just over 28 s to compute. A further 60 s of wall clock time were required to sort the final output for concrete assignment. Sorting optimizations were not investigated, though they contributed more than 68% of our overall solution runtime.

Table 2 Distribution of VMs across seven equivalence sets from a representative large-scale experiment.

VM equivalence set name	Concrete VM count
Micro	2,000
Small	2,000
Medium	2,000
Large	1,000
XL	500
DXL	250
QXL	100

In both experiments, we chose a distribution more heavily weighted towards the virtual machine sizes that the author has encountered during many years of professional and academic creation of VMs for cloud computing. Based on the author's own practical experience in corporate VM creation and management for over a decade, it is apparent that many virtual machines begin their existence as a proof-of-concept system, generally sized as small as possible thus many are simply sized small at creation time. Further, we chose this size skew because virtual machines that require high levels of resources such as RAM or CPU are often selected for migration to dedicated hardware (often as the result of latency or performance reasons). Likewise, we argue that virtual machine-based data center operators, do not tend to create data centers consisting only of massively over-configured hosts. Thus, our experiments were targeted along a set of experiments fitting with our own empirical observations of cloud data centers. We should note however that our simulator does not stipulate any requirements on the number of VMs simulate or the nominal host capacities represented, thus more extreme simulations can indeed be performed. An example of the configuration file used to perform the experiments can be seen in Appendix. This configuration file format has proven to be expressive enough to run a variety of simulations during the course of this research.

Our approach is unique in that it is based on an algorithm with complexity and runtime growth governed by the number of hosts and VMs, but substantially mitigated using an intermediate partial solution that is governed by the number of distinct groupings of equivalent sized virtual machines (the classifications of virtual machine types) used in the system. While this algorithm may indeed break down for exceedingly large numbers of hosts (perhaps 500 or 1,000 hosts), we maintain that our goals are for the autonomous management of small to medium sized data centers that we consider to be well in the operational range of our solution. We have not simulated data centers beyond the maximum configuration reported here. This is in part because there will always be some degree of reservation regarding our use of simulation software rather than real data center hardware, rendering further experimentation subject to those same criticisms. We submit that VM management research should not be the bastion of private industry researchers with privileged access to that exorbitantly expensive class of hardware and production virtual machines and did what we could with the resources

available to us. Secondly, we limited the reach of our simulations because we have no empirical data sets for sizing the virtual machine distribution across a data center that size beyond those that we reported. Unsurprisingly, various cloud virtual machine providers rejected a plurality of requests for that sort of information. Not only has targeting this class of data center proved to be computationally tractable from a fully autonomous management perspective, but this class of data center are those that are much less likely to opt for exorbitant hardware capital expense needed to maintain performance while achieving consolidation (i.e., they want to consolidate to save money, and are generally less equipped with operational staff to manually check each migration operation if using live migration based consolidation. Effectively, this class of user and data center are the users who need a lightweight, autonomous solution the most.

Formal comparisons to other VM-based cloud management systems are difficult in this type of research because this work was simulated while other published results may be run on other proprietary simulators or on real hardware. Additionally, it is notable that systems that perform bin packing of VMs do not generally open source or otherwise license their source code for no cost, and thus requiring black box reimplementation of described algorithms in order to perform meaningful comparisons. Even under clean room implementation cases, one is never sure if the original authors used tweaks or modifications etc., which would render a clean room implementation comparison meaningless.

With respect to published values of VM's packed as a function of time taken to perform the packing, we present our results as compared to those of *Ferdaus et al. (2014)* in their work using an Ant Colony Optimization based on vector algebra. Their experiments were performed on a smaller scale but roughly comparable server with respect to single core CPU speed. Our approach is not threaded, and we were unable to determine from a careful reading of their paper if their solution was. Their work is presented, similarly to our own, by using simulation. While they solve bin packing in a completely different approach to ours, they find their results favorable when compared to approaches like genetic algorithms. However, this improvement upon, or parity with their work can be considered a reasonable benchmark of success based on the literature. Their results extrapolated from their Fig. 2 along with their surrounding text would at first glance appear much better than our own. We readily admit that our solution seemingly performs much worse for small-scale experiments as described in our Table 1. As evidence of this, consider that our own packing solution for 78 virtual machines took 22 s in total while their comparable results reportedly took close to 0 s to compute. Their results are surely impressive for these small cases! However, it is our opinion that for these small cases of managed VMs, either of our algorithms might be suitable for effectively continuous online application. However, their largest scale result includes only 2,000 virtual machines and took "around 25 s on average." In contrast, our largest simulation results involved 7,850 VMs and completed in 88 s. The only viable way to compare our results is to extrapolate by optimistically assuming their results scale linearly in the number of VMs (which Ferdaus et al. explicitly admit is not the case stating "it is observed that computation time increases non-linearly with the number of VMs with small gradient"). Using this

optimistic extrapolation, we paint their work in the best possible light and attempt to perform a meaningful comparison to our own work. If we assume their results suggest a management rate of 80 VMs/sec (derived from their published values reporting 2,000 VMs managed in 25 s) their algorithm on a 7,850 VM problem set would yield around 98 s run time that would hypothetically, but optimistically, exceed our own by 11% (or 18 s). While these numbers do not provide conclusive superiority of one technique over the other, they do seem comparable. Their analysis includes remarks that their results perform better than FFD based approaches, and a sizeable portion of our concrete assignment computation is based on a type of FFD derived technique which may explain much of their performance benefit especially in the scenarios with a small number of managed VMs. In addition, we submit that their solution only considers, ''CPU, memory, and network I/O as relevant server resources in the context of VM consolidation'' while ours considers CPU, memory, aggregate network I/O as well as cumulative network read I/O and cumulative network write I/O as well as network read/write/aggregate I/O. In addition we support preferential colocation and anti-colocation constraints that their solution does not handle at present. In summation of this section, our work performs comparably when extrapolating to the results of *Ferdaus et al. (2014)* in as favorable a means as possible while our reported results handled more resource dimensions, and included arguably necessary resources like colocation and anti-colocation for viable production systems. Lastly our work may be seen as preferable for production deployment based on the deterministic nature of our solution while the approach taken by Ferdaus et al. is probabilistic.

We conclude our comparative analysis to other known works in the field by comparing our work to a recent paper from *Hwang & Pedram (2013)*. While they also take a hierarchical approach to virtual machine consolidation, using simulation for their published results, we note the following differences between our respective works. First, Hwang and Pedram decompose the problem in a slightly different way, opting for decomposition with a global manager that places VMs into clusters or functional groups, along with a local manager that assigns VMs to hosts within a given cluster. Our highest-level hierarchy is grouping VMs not based on cluster functionality but rather on their instance type or equivalence set. The second major difference of note is that their approach is heuristic in nature and appears to be based on a modified FFD algorithm that considers an aggregate of the free resources available as the term for greedy selection. Our work is hierarchical in that once a generic instance-type allocation has been prescribed, the concrete assignment phase hierarchically uses a type of FFD that descends through an *a priori* ordered list of resource dimensions and colocation/anti-colocation criteria to be used as tie-breakers for greedy selection. Thus, the FFD greedy selection criterion hierarchy is only used as needed. Their work seemingly does not consider colocation or anti-colocation considerations. Comparing to their results, we can rely on the single published reference of concrete runtimes, Fig. 9 in their work. Note the remaining figures they have included are presented as normalized runtimes for comparative purposes. Through email correspondence, the author of this work has confirmed the label in their Fig. 9 is incorrectly identified with a label of ''seconds,'' when it should in fact be labeled

"microseconds". Referring concretely to their Fig. 9, the best-case run time for packing management of 2,000 VMs is $\sim 0.6 * 10^7$ microseconds (6 s) using a window size of 0. Their worst case published run time with a window size for the management of 2,000 VMs is 22 s (0.4 min). As you will recall from earlier in this section, our solution for a VM packing problem that was four times as large (8,000 VMs) took under two minutes. Once more we extrapolate using an optimistic linear extrapolation to a problem size of 4× and we estimate optimistically that Pedram and Hwan's solution for a comparable problem size would take 24 s in the best case. It is is evident that a linear extrapolation is generous for comparative purposes as their results show a decidedly non-linear run-time increase covering 500, 1,000, 1,500, and 2,000 VM cases varied across each of their sliding window size selections. In their worst-case results (sliding window size of 60) using the same optimistic 4× extrapolation, we optimistically conclude an estimated lower-bound runtime would be approximately 88 s (which is, coincidentally, our empirically measured value for 7,850 VM problem set). We conclude that our algorithm performs slower than Pedram and Hwang's best-case published solution when extrapolated optimistically for the ∼8,000 VM problem size, but performs comparably to their worst-case solution when extrapolating though our solution considers location constraints that theirs does not. Accounting for generous extrapolations, we expect our solution would perform comparably, if not better, in practice for larger problem sizes while accounting for colocation and anti-colocation that their solution seemingly does not handle.

CONCLUSIONS AND NEXT STEPS

Our results show that decomposing the problem of virtual machine consolidation in cloud computing data centers into an abstract solution phase and a concrete fulfillment stage is advantageous in that it is easy to implement, allows satisfaction of multiple resource constraints, provides an easy means for implementing oversubscription support, as well enabling a straightforward approach to colocation and anti-colocation requirements with very reasonable runtimes (under 90 s) for moderate numbers of virtual machines (7,850 VMs). In this way we can make use of observations data, if available, during the second phase of concrete assignment to improve placement decisions. These results compare favorably to the published literature from Ferdaus et al. as well as the work of Hwang and Pedram. However we tender these comparisons loosely, given the different motivations of the work, as well as the different problem set sizes used in the various publications and the requisite need to extrapolate in our effort for meaningful comparison. Further, it is apparent that our ongoing work should investigate potential sorting optimization to ameliorate the sorting activity that dominated our runtime profile.

While the work of *Roytman et al. (2013)* demonstrated one means for performance characterization of potential consolidation impacts by using a staging area for new virtual machines, our research group believes that online detection of poor placement or colocation decisions should be a separate process architecturally. Furthermore, we believe the process of VM placement will eventually move to a fully dynamic and continuous

system that operates at the data center scale. Live-guest migration and fast networking technologies are beginning to show the promise of extremely short and non-disruptive live-migrations for even large-scale enterprise sized virtual machines.

We are therefore working on an on-line, machine-learning-based (*Dow, 2015*) system to perform the task of detecting poor colocation decisions as a remedial effort for continuous cloud-scale VM rebalancing. We believe this work will complement the rapid consolidation planning implementation we have outlined here. It is our assertion that evidence-based VM continuous placement adjustment decisions will become the nominal operational reality for IaaS virtual machine clouds.

The desire to pack VM's expediently stems from implications regarding server power consumption in heterogeneous compute clouds. *Fan, Weber & Barroso (2007)*, describe the covariant relationship between CPU usage and server power consumption as a linear relationship between power consumption of the server and the increase of CPU utilization. The range of this covariance extends from virtually idle through fully utilized states.

$$P(u) = P_{\text{idle}} + (P_{\text{busy}} - P_{\text{idle}}) * u. \tag{4}$$

Equation (4) shows a linear model of power estimation based on CPU and idle power consumption. In Eq. (4), P is the estimated power consumption, while P_{idle} is the power consumption of an idle server, and with P_{busy} as the power consumption of a fully utilized server. The value u is the instantaneous present CPU utilization of the host. Their work demonstrates that using this simple linear model, one can estimate the future power consumption of a server with error below 5%. They further proposed empirical, nonlinear, model is as shown in Eq. (5):

$$P(u) = P_{\text{idle}} + (P_{\text{busy}} - P_{\text{idle}})(2u - u^r). \tag{5}$$

Equation (5) shows an improved empirical model of power estimation based on calibration experimentation. In Eq. (5), the parameter r is an experimentally obtained calibration parameter that minimizes the square error for the given class of host hardware. To use this formula, each class of host hardware must undergo a set of calibration experiments to generate the value of r that tunes the model. Their work, which spans thousands of nodes under different types of workloads, demonstrates a prediction error below 1% for a tuned empirical model.

Using the results from Fan et al., it may be possible to treat hosts as equivalent resources in the same fashion as virtual machines are considered equivalent in this work. It is interesting to note that the previously mentioned work of Ferdaus et al. also uses the same general formulation for modeling power consumption in their Fig. 9 as shown above. This lends some degree of consensus to the adoption of this relatively straightforward approach for modeling power consumption of virtual machines. The approach envisioned makes the assumption that one could use the "smallest" hardware resource capacity vector from a set of hosts that have been arbitrarily clustered, perhaps through K-means or similar clustering approaches) by their power consumption

estimation formulas (Eqs. (4) and (5) in this work). This work seems a logical follow on and a reasonable next step to investigate.

APPENDIX: SIMULATION CONFIGURATION FILE

HARDWARE

```
# The total number of physical host servers to be modeled
MAX_SIMULATED_SERVERS=255

#The largest server RAM allocation present in the cloud (Largest server ram in
MB)
MAX_SERVER_RAM_SIZE=524288

#The largest smallest RAM allocation present in the cloud (Smallest server ram
in MB)
MIN_SERVER_RAM_SIZE=128

# The largest server CPU allocation present in the cloud (Number of hardware
cores)
MAX_SERVER_CPU_SIZE=124

#The largest smallest RAM allocation present in the cloud (Smallest server ram
in MB)
MIN_SERVER_CPU_SIZE=1

# The largest server NET allocation present in the cloud (in GB/sec)
MAX_SERVER_NET_SIZE=10

# The largest server NET RX allocation present in the cloud (in GB/sec)
MAX_SERVER_NET_RX_SIZE=10

# The largest server NET TX allocation present in the cloud (in GB/sec)
MAX_SERVER_NET_TX_SIZE=10

# The largest server IO allocation present in the cloud (in GB/sec)
MAX_SERVER_IO_SIZE=10

# The largest server IO READ allocation present in the cloud (in GB/sec)
MAX_SERVER_IO_READ_SIZE=10

# The largest server IO WRITE allocation present in the cloud (in GB/sec)
MAX_SERVER_IO_WRITE_SIZE=10
```

OVERCOMMIT

```
# Overcommit is the oversubscription or overallocation of resources to virtual
machines
# with respect to the physical hosts maximum allocation of a particular
resource.
#
```

```
# As an example, setting MEMORY overcommit to 1 means a 1:1 ratio of allowed
cumulative
# vm memory consumption to the hosts physical ram.
#
# Setting a CPU overcommit value of 2 indicates scheduling up to 2 virtual cpu
cores
# per single physical cpu core on the host.
#
# NOTE: Values should always be 1 or greater...

MEM_OVERCOMMIT_RATIO=1

CPU_OVERCOMMIT_RATIO=1

NET_OVERCOMMIT_RATIO=1

NET_RX_OVERCOMMIT_RATIO=1

NET_TX_OVERCOMMIT_RATIO=1

IO_OVERCOMMIT_RATIO=1

IO_READ_OVERCOMMIT_RATIO=1

IO_WRITE_OVERCOMMIT_RATIO=1
```

VIRTUAL MACHINES

```
# The number of MICRO Sized Virtual Machines
MICRO=30

# The number of SMALL Sized Virtual Machines
SMALL=20

# The number of MEDIUM Sized Virtual Machines
MEDIUM=10

# The number of LARGE Sized Virtual Machines
LARGE=10

# The number of EXTRA LARGE Sized Virtual Machines
XL=5

# The number of DOUBLE EXTRA LARGE Sized Virtual Machines
DXL=3

# The number of QUADRUPLE EXTRA LARGE Sized Virtual Machines
QXL=1
```

MIGRATION PLANS

```
# The maximum possible inbound migrations per host (for bandwidth limiting or
cpu concerns with parallel migrations)
```

```
MAX_CONCURRENT_INBOUND_MIGRATIONS_PER_HOST=2

# The maximum possible outbound migrations per host
# (for bandwidth limiting or cpu concerns with parallel migrations)
MAX_CONCURRENT_OUTBOUND_MIGRATIONS_PER_HOST=2

# The maximum possible cumulative inbound and outbound migrations per host
# (for situations where the above alone will not tell the whole story of
hypervisor impact)
MAX_CONCURRENT_OVERALL_MIGRATIONS_PER_HOST=3
```

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

The author received no funding for this work.

Competing Interests

Eli M. Dow is an employee of IBM Research.

Author Contributions

- Eli M. Dow conceived and designed the experiments, performed the experiments, analyzed the data, contributed materials/analysis tools, wrote the paper, prepared figures and/or tables, performed the computation work, reviewed drafts of the paper.

Data Availability

The following information was supplied regarding data availability:

Research performed in support of this publication did not generate appreciable or generally useful raw data beyond the comparative summary statistics reported herein.

REFERENCES

Campegiani P. 2009. A genetic algorithm to solve the virtual machines resources allocation problem in multi-tier distributed systems. In: *Second international workshop on virtualization performance: analysis, characterization, and tools (VPACT 2009)*. *Available at http://www.paolocampegiani.it/vpact.pdf* .

Campegiani P, Lo Presti F. 2009. A general model for virtual machines resources allocation in multi-tier distributed systems. In: *Autonomic and autonomous systems, 2009. ICAS'09*. Piscataway: IEEE, 162–167.

Dow EM. 2015. Inciting cloud virtual machine reallocation with supervised machine learning and time series forecasts. In: *Proceedings of the enterprise compute conference (ECC)*. *Available at https://ecc.marist.edu/conf2015/pres/DowIncite-Presentation-ECC2015.pdf* .

Dow EM, Matthews N. 2015. Virtual machine migration plan generation through A∗ search. In: *Cloud networking (CloudNet), 2015 IEEE 4th international conference on cloud networking*. Piscataway: IEEE, 71–73.

Fan X, Weber W, Barroso LA. 2007. Power provisioning for a warehouse-sized computer. In: *Proceedings of the 34th annual international symposium on Computer architecture*. New York: ACM, 13–23.

Ferdaus MH, Murshed M, Calheiros RN, Buyya R. 2014. Virtual machine consolidation in cloud data centers using ACO metaheuristic. In: *Euro-par parallel processing*. New York: Springer, 306–317.

Gandhi A, Harchol-Balter M, Das R, Lefurgy C. 2009. Optimal power allocation in server farms. In: *Proceedings of ACM SIGMETRICS. Available at http://www3.cs.stonybrook.edu/~anshul/sigmetrics_2009_tech.pdf*.

Greenberg A, Hamilton J, Maltz DA, Patel P. 2009. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review* **39(1)**:68–73 DOI 10.1145/1496091.1496103.

Gu J, Hu J, Zhao T, Sun G. 2012. A new resource scheduling strategy based on genetic algorithm in cloud computing environment. *Journal of Computers* **7(1)**:42–52 DOI 10.4304/jcp.7.1.42-52.

Hwang I, Pedram M. 2013. Hierarchical virtual machine consolidation in a cloud computing system. In: *Proceedings of the 2013 IEEE sixth international conference on cloud computing (CLOUD'13)*. Piscataway: IEEE Computer Society, 196–203.

Isci C, Liu J, Abali B, Kephart JO, Kouloheris J. 2011. Improving server utilization using fast virtual machine migration. *IBM Journal of Research and Development* **55(6)**:365–376.

Jiang Y, Shen X, Chen J, Tripathi T. 2008. Analysis and approximation of optimal co-scheduling on chip multiprocessors. In: *Proceedings of the seventeenth international conference on parallel architectures and compilation techniques*, 220–229.

Jiang Y, Tian K, Shen X. 2010. Combining locality analysis with online proactive job co-scheduling in chip multiprocessors. In: *Proceedings of the international conference on high-performance embedded architectures and compilers*, 6–8.

Johnson DS, Demers A, Ullman JD, Garey MR, Graham RL. 1974. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing* **3(4)**:299–325 DOI 10.1137/0203025.

Kohavi R, Henne RM, Sommerfield D. 2007. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In: *Proceedings of the thirteenth annual ACM SIGKDD international conference on knowledge discovery and data mining*. New York: ACM, 959–967.

Lynar TM, Herbert RD, Simon S. 2009. Auction resource allocation mechanisms in grids of heterogeneous computers. *WSEAS Transactions on Computers* **8(10)**:1671–1680.

Mars J, Tang L, Hundt R, Skadron K, Soffa ML. 2011. Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations. In: *Proceedings of the forty-fourth annual IEEE/ACM international symposium on microarchitecture (Micro44)*. New York: ACM.

Narayanan D, Donnelly A, Rowstron A. 2008. Write offloading: practical power management for enterprise storage. In: *Proceedings of the file and storage technologies conference*, San Jose, CA, 253–267.

Panigrahy R, Talwar K, Uyeda L, Wieder U. 2011. Heuristics for vector binpacking. Microsoft Research Technical Report. Redmond, Microsoft. *Available at http://research.microsoft.com/apps/pubs/default.aspx?id=147927*.

Roytman A, Kansal A, Govindan S, Liu J, Nath S. 2013. PACMan: performance aware virtual machine consolidation. In: *Proceedings of the 10th international conference on autonomic computing (ICAC)*, 83–94. *Available at https://www.usenix.org/conference/icac13/technical-sessions/presentation/roytman*.

Schurman E, Brutlag J. 2009. Performance related changes and their user impact. In: *O'REILLY: web performance and operations conference (velocity)*.

Tian K, Jiang Y, Shen X. 2009. A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In: *Proceedings of the ACM international conference on computing frontiers*. New York: ACM.

Tulloch M. 2010. Achieve higher consolidation ratios using dynamic memory. BizTech Article. *Available at http://www.biztechmagazine.com/article/2010/12/achieve-higher-consolidation-ratios-using-dynamic-memory*.

United States Environmental Protection Agency. 2007. Report to congress on server and data center energy efficiency public law 109-431. Technical Report, EPA ENERGY STAR Program. Washington, D.C., EPA.

VMware, Inc. 2006. The role of memory in VMware ESX server 3. Palo Alto: VMware, Introduction section, paragraph 1, page 1. *Available at https://www.vmware.com/pdf/esx3_memory.pdf*.

Zhang D, Ehsan M, Ferdman M, Sion R. 2014. DIMMer: a case for turning off DIMMs in clouds. In: *ACM symposium on cloud computing (SOCC)*. New York: ACM.