Saving the planet with bin packing - Experiences using 2D and 3D bin packing of virtual machines for greener clouds

Thomas Hage
University of Oslo: Institute of Informatics
Oslo, Norway
thomhage@student.matnat.uio.no

Kyrre Begnum, Anis Yazidi
Oslo and Akershus University College
of applied sciences
Oslo, Norway
{*kyrre.begnum—anis.yazidi*}*@hioa.no*

***Abstract*—Greener cloud computing has recently become an extremely pertinent research topic in academy and among practitioners. Despite the abundance of the state of the art studies that tackle the problem, the vast majority of them solely rely on simulation, and do not report real settings experience. Thus, the theoretical models might overlook some of the practical details that might emerge in real life scenarios. In this paper, we try to bridge the aforementioned gap in the literature by devising and also deploying algorithms for saving power in real-life cloud environments based on variants of the 2D/3D bin packing algorithms. The algorithms are tested on a large OpenStack deployment in use by staff and students at Oslo and Akershus University College, Norway. We present three different adaptions of 2D and 3D bin packing, incorporating different aspects of the cloud as constraints. Our real-life experimental results show that although the three algorithms yield a decrease in power consumption, they distinctly affect the way the cloud has to be managed. A simple bin packing algorithm provides useful mechanism to reduce power consumption while more sophisticated algorithms do not merely achieve power savings but also minimize the number of migrations.**

***Keywords*-Cloud computing, Green IT, Bin packing, Real-life Experiences**

I. Introduction

The role of system administrators today is not only to provide data centers uninterrupted power, better security, extension opportunities and availability, but also to operate the cloud in a way that efficiently optimizes energy. Most research in this field base their results on simulations or test environments. Very rare is the reported research that attempts power saving for large cloud environments based on any of the major private cloud solutions, like OpenStack, Eucalyptus, OpenNebula or CloudStack. What is it like to run a scaling cloud in production? What practical details are overlooked by the theoretical models?

A. Related work in power saving

We will proceed to providing a brief overview of related work. The VirtualPower[6] project explores the possibility to integrate online power management mechanisms and policies in a virtualized environment. The goal of this project is to efficiently manage workloads through migration technology to increase power efficiency. The implementation was done in Xen hypervisor. Dhiman et al. [1] resort to Vgreen, a multi-tier software tool to render virtual machine management in a clustered virtualized environment more energy efficient. The system was implemented on a Xen hypervisor and improved both average performance and system-level energy savings by 40 percentage. In [7], the authors focus their study on the reduction of electricity consumption through dynamic voltage frequency scheduling (DVFS). Zhang et.al [9] used element from the theory of Model Predictive Control(MPC) to find the optimal control policy for dynamic capacity provisioning. The solution aims to find a trade-off between energy savings and capacity re-configuration cost. The framework is an initial step towards building a full-fledged management system. Bin packing is a well studied problem NP hard problem that has found applications in cloud computing for power saving by live migration possibilities. Btrplace[2] is a flexible consolidation manager for highly available applications. This is a manager for resources in data centers which dynamically consolidate workloads. Btrplace provides administrators the ability to give virtual machines placement constrains and requirements to gain the best performance. Btrplace was tested on a simulated datacenter containing 5000 servers hosting 30,000 VMs, and it could find a viable configuration in under 3 minutes. Space defragmentation heuristic for 2D and 3D bin packing problems [10] illustrates a technique to make the packing process more efficient by combining small unused gaps in bins or containers to make room for more objects in each container. The project presents both 2 dimensional and 3 dimensional bin packing algorithms.

II. Model overview and description of the algorithms

In this section, we present three different bin packing algorithms that are subsequently implemented and tested in OpenStack. In this section we present the terminology used in the algorithm and present each in turn. In the next section, we discuss their implementations and results.

A. Applying bin packing to virtual machines

When applying bin packing to a cloud, the compute nodes (i.e the hypervisor servers containing the virtual machines) represent the bins in which to pack the virtual machines in such a way that we need as few bins as possible. The "space" of a bin, in which to pack virtual machines is defined by several constraints. The constraints can be arbitrary as long as they are reduced as virtual machines are placed in the bins. The number of constraints are also the dimensions of the bin. For example, the three constraints of a normal box are height, width and depth. For a compute node it may be CPU and memory or more. Bin packing is a well known NP complete problem, this means that for real life adaption, one has to choose algorithms that don't guarantee the most optimal solution, but close to optimal. In the design of a solution we are faced with two choices: the number and type of constraints and the algorithm for looking for the best solution. Let C_n denote the nth compute node in a cloud environment. Virtual CPUs and memory are two obvious constraints of a hypervisor and are written as C_{n_VCPU} and C_{n_MEM} in capitals respectively. The current use of a constraints is written in non-capitals, like C_{n_vpcu}. The ith virtual machine is represented as V_i. Every VM will consume an amount of the aforementioned constraints of a bin, such as VCPU and memory resources, denoted as V_{i_VCPU} and V_{i_MEM}. These two constraints are common for all three algorithms, however algorithm II and III add an additional constraint.

B. Algorithm I: Simple 2D first fit bin packing

The first algorithm we present is the most straight forward and presents the fundamentals of the process. Consider an array of N running compute nodes. We want to empty one and one compute node, giving it the name C_e. The compute node currently eligible for receiving the virtual machine is C_r. We start at the high end of the compute nodes, $C_e = C_N$, and try to find a place for each virtual machine starting at the other end $C_r = C_1$. As soon as we find a compute node with space, we place the VM there and pick the next VM on C_e and start at $C_r = C_1$ again. As soon as the compute node C_e is emptied it can be shut down. The algorithm then proceeds to $C_e = C_{N-1}$ and so on. If we find that we are trying to place a virtual machine into the same compute node, $C_e == C_r$, we have reached the end as there is no room below in the array.

In pseudocode, the algorithm can be described like this:

```
FOR C_e in ( C_N, C_{N-1} ... C_1 ) DO
  FOREACH VM_e in C_e DO
    FOREACH C_r in (C_1, C_2, ... C_e) DO
      IF C_r == C_e THEN
        Finished()
      END
      IF (C_r_mem + V_e_mem ≤ C_r_MEM) and
         (C_r_vcpu + V_e_vcpu ≤ C_r_VCPU) THEN
           MigrateVM(VM_e,C_r)
           LAST
      END
    END
  END
  SHUTDOWN(C_e)
END
```

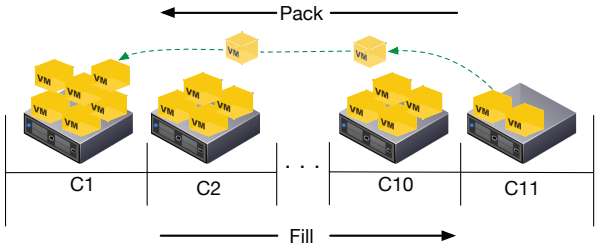

Figure 1. *The first fit algorithm starts from the end of the compute node array and looks for space from the other end.*

There are obvious shortcomings with this algorithm, as it may create gaps in some of the compute nodes as it only considers one and one VM. Also, it does not take the current placement much in account. For instance, if C_N happened to have the most virtual machines and C_1 the least, it would be better to go the other way in order to reduce the number of live migrations. Lastly, and most importantly, all virtual machines will be treated equal and will be crammed into as few compute nodes as possible. This may severely affect the performance of all the virtual machines.

C. Algorithm II: 2D best fit bin packing with CPU zones and minimal migrations

With this algorithm we want to address the two biggest concerns from the previous one: reducing the number of potential live migrations and allowing some form of performance barrier so we may ensure the performance of some virtual machines. Every compute node now has split its CPUs into two sets: low quality and high quality. This is implemented using CPU affinity settings on the compute nodes for each virtual machine. A virtual machine that is classified as non-important will get the CPU affinity belonging to the low quality set while the important virtual machines are placed int he high quality set. As these two sets don't overlap, we basically get two separate constraints C_{n_LQVCPU} and C_{n_HQVCPU}, but a VM can only reduce the number in one of them. In effect, this becomes two 2D spaces with overlapping constraints.

The constraints on the two CPU dimensions can be adjusted based on the desired amount of overbooking. For example, consider a compute node of 64 physical CPUs. Let the high quality set span from CPU 1 until 24, and the low quality set from 25 until 60, leaving the compute node itself with 4 dedicated CPUs. Since the low quality set should allow for a higher utilization of space, we can set the overbooking factor to 8:1, resulting in $C_{n\_LQVCPU} = 35{*}8 = 280$. For the high quality set we can use 2:1 and get:

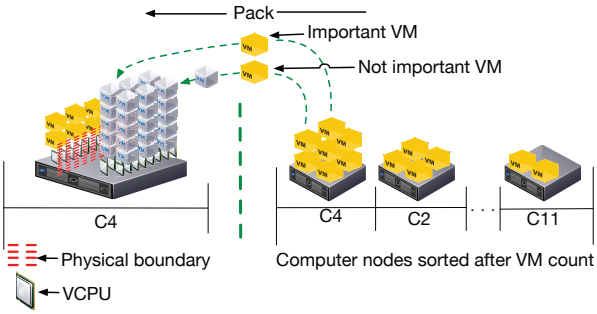

Figure 2. *This algorithm reduces the number of live migrations as it will begin packing from the least populated compute node to the most populated one.*

$C_{n\_LQVCPU} = 24 * 2 = 48$. In order to reduce the number of migrations, we now traverse the array of compute nodes from the least populated to the most populated compute node. In pseudocode, the algorithm can be described like this:

```
FOR C_e in SORT('VM_COUNT','ASC',( C_N, C_{N-1} ... C_1 )) DO
  FOREACH VM_e in C_e DO
    FOREACH C_r in SORT('VM_COUNT','DESC',( C_N, C_{N-1} ... C_1 )) DO
      IF C_r == C_e THEN
        Finished()
      END
      IF isImportant(VM_e) THEN
        IF (C_r_mem + V_e_mem ≤ C_r_MEM) and
           (C_r_hqvcpu + V_e_vcpu ≤ C_r_HQVCPU) THEN
            MigrateVM(VM_e,C_r)
            LAST
        END
      ELSE
        IF (C_r_mem + V_e_mem ≤ C_r_MEM) and
           (C_r_lqvcpu + V_e_vcpu ≤ C_r_LQVCPU) THEN
            MigrateVM(VM_e,C_r)
            LAST
        END
      END
    END
  END
  SHUTDOWN(C_e)
END
```

The classification of what virtual machines are important is outside of this algorithm, but in our case we let it be something the tenant would choose. Combined with a future billing system that would take this into account, the user would have an incentive to classify unused virtual machines as not important for a period of time. One could also detect importance by looking for special flavors (e.g instance types in Amazon), but that would render a virtual machine important for it's entire lifetime, which is impractical. On the other hand, machine learning approaches could attempt at this classification, like the learning model presented in the Dependable Virtual Machine Placement project[8].

D. Algorithm III: 3D bin packing with importance as constraint

The last algorithm we present in this paper is based on seminal work of Martello et al.[4] for solving the three-dimensional bin packing algorithm. The algorithm resorts to a branch-and-bound optimization and was shown to exhibit fast convergence and to achieve near-optimal solution. To adapt the algorithm to our particular VM packing problem, we chose VCPU and memory to be the first two of the constraints. For the third constraint, we define a novel concept called importance capacity. Every virtual machine will have an importance weight associated with it. When considering the dimensions for 3D bin packing they together create a much larger space in which to place virtual machines. Virtual machines can be placed next to each other in a 3D space, allowing them to "occupy" the same resource constraint multiple times. This is ultimately the challenge when transforming something that uses resources to something that uses space.[5] One therefore has to chose the dimensions with care. Using $VCPU * MEM * IMPORTANCE$ directly as dimensions would translate to $64 * 256 * 10$ which would fit a very high number of virtual machines. In the experimental settings, we assign a weight 2 to an important virtual machine and 1 to a non important virtual machine. Intuitively speaking, the third constraint, importance capacity, permits to restrict the number of important machines co-existing in the same physical machines. The constraint is motivated by quality of service consideration in Cloud Computing. In fact, it is known that two VMs residing on the same physical machine and sharing the same physical CPU can interfere with each other resulting in a lower quality of service experienced by both VMs.

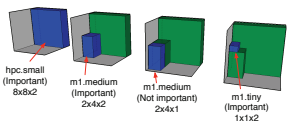

Figure 3. *This algorithm calculates a volume optimized solution regardless of previous location.*

For example, consider an important VM with 2 VCPUs, 4GB of memory. This would correspond to the VM being a cuboid of size $2 * 4 * 2$, taking up a volume of 16. In a bin of $8_{VCPU} * 8_{MEM} * 6_{IMPORTANCE}$ we would be able to fit 24 of them (or 48 non-important ones of the same type). For our compute nodes, $8 * 8 * 6$ was chosen as this best reflected the capacity we wanted.

Two heuristic algorithms are used called H_1 and H_2, according to the descriptions given in Martello et al[3]. The first algorithm H_1 constructs number of layers of dimensions $W \times H \times d, (d \leq D)$ which in our case is VCPU, memory and importance. By solving a bin packing algorithm with one-dimension defined by the depth of the layers, the result is full bins of depth D. Algorithm H_2 fills every bin to maximize the volume filled. The result is a list containing a volume optimized solution for placement of virtual machines.

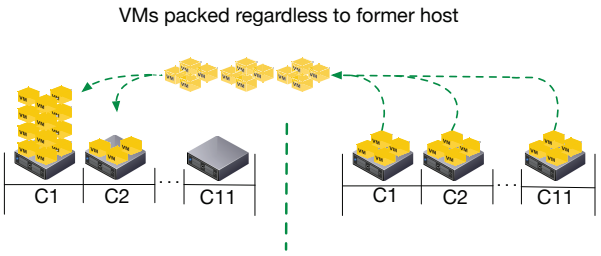

Figure 4. *This algorithm calculates a volume optimized solution regardless of previous location.*

III. Implementation and short term results

All three algorithms were implemented as separate scripts and tested on the ALTO cloud, an OpenStack cloud consisting of 11 compute nodes with a combined capacity of 704 physical cores and 3TB of memory for virtual machines. At the time of testing, the number of virtual machines in use in ALTO was in the hundreds. Like most cloud environments, the virtual machines were mostly idle and as a result, the compute nodes tended to have a stable energy usage of 270W each.

The scripts collected data from the central database and used that to calculate the optimal placements. One difference in the scripts as opposed to the pseudocode, was that virtual machines were not migrated directly, but scheduled for migration. This allowed us to review the resulting migrations first for safety. Once the scripts were tested sufficiently, we let them conduct the migrations at the end of the script automatically. The compute nodes were managed through a separate console interface, allowing us to collect power usage and to power on compute nodes that were shut down. OpenStack did not provide an interface to classify important VMs and non-important ones. In order not to interfere with the tenants, we defined the non important virtual machines in a separate file so that by default any VM is important unless otherwise stated. Before every test, we recorded the current placement of each virtual machine so we could set everything back to it's original place after each experiment. For a comparison, we recorded the placement of the virtual machines at one point and tested all three algorithms with variations in the parameters. After each test, the virtual machines were migrated back to their original position, resetting the scenario. A total of 198 virtual machines were active at the time of the experiments. Table 1 shows the different results achieved from the algorithms. Algorithm I was tested with a low VCPU capacity, only 64, which was a 1:1 relationship to the physical cores. The second version uses the 16:1 factor that is also the default in OpenStack. For Algorithm II and III, 75 of the virtual machines were classified as not important. Figure 4 illustrates what happens when algorithm II is executed on the cloud with no previous packing. The cloud, using 11 compute nodes at the time, is reduced to 7 compute nodes. The algorithm executes in a matter of seconds, however the virtual machines are migrated one by one, taking about 30 minutes to complete. Note, that this is a scenario where no previous packing has been performed. If the number of virtual machines changes little until the next execution, few if any migrations will take place. The compute nodes were shut down after all migrations were complete. This, also, was for concerns that there might be a erroneous situation and all actions should be postponed until the desired state was verified. Consider that in a balanced environment, we would normally have about the same amount of virtual machines on every compute node, especially if they are similar in resource usage. It is therefore normal that algorithm I and II should perform almost equal in terms of the number of migrations as it does not matter where we start migrating from. What is different is that algorithm II separates important from not-important virtual machines providing a guarantee that too many important virtual machines randomly get packed on the same compute node.

IV. Long term testing

Algorithm II was selected for long-term testing over 3 weeks due to the possibilities to lower the amount of live migrations and the ability to determine the difference between important and not important virtual machines. There is mostly important virtual machines in the environment and all virtual machines are given one-to-one VCPU. The environment is filled with virtual machines doing experiments for other master thesis so there is no overbooking. When the policy was started there where around 80 virtual machines, but within the first week the number of virtual machines raised to 150. The policy will be executed every hour, every day all week for three weeks. The long term policy takes the number of spare physical machines as an argument which allows the padding to be different according to night and day. In the design phase of this project there where suggested that the threshold and buffer should be calculated in percent like this where T_{VCPU} is the available space:

$$1 - \frac{U^t_{VCPU}}{C_{tn}} > T_{VCPU} \quad (1)$$

The policy first started to calculate new locations to optimize the space available at the nodes which are sorted according to number of virtual machines. After all calculations is done the live migrations of the virtual machines is conducted. These are illustrated at the top right graph. It took about 20 minutes to live migrate all virtual machines to new locations. After all virtual machines had gotten new locations the policy took down 5 compute nodes. Compute05 is one of the compute nodes which received lots of virtual machines and it could be interesting to see the power consumption when it received 20 virtual machines. In figure

Algorithm tested:	**Algorithm I**		**Algorithm II**		**Algorithm III**	
	$C_{n\_VCPU} = 1:1$	$C_{n\_VCPU} = 16:1$	$C_{n\_HQVCPU} = 8:1$ $C_{n\_LQVCPU} = 16:1$	$C_{n\_HQVCPU} = 1:1$ $C_{n\_LQVCPU} = 16:1$	$C_{n\_HQVCPU} = 2:2$ $C_{n\_LQVCPU} = 2:1$	$C_{n\_HQVCPU} = 3:3$ $C_{n\_LQVCPU} = 3:1$
Number of virtual machines	198	198	198	198	198	198
Resulting Compute nodes	6	3	7	6	6	8
Live Migrations	92	160	129	86	189	179
Power consumption reduction	45%	72%	63%	45%	45%	27%

Table I
The three algorithms tested on the same placement of virtual machines. A total of 198 virtual machines were running.

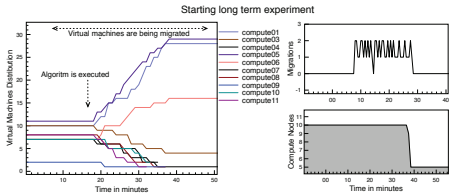

Figure 5. *Graph illustrating the startphase of the long term test of policy 2, all compute nodes to the left, live migrations top right and compute nodes active bottom right*

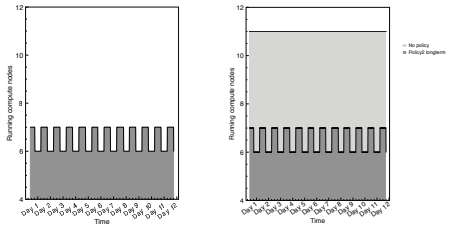

Figure 7. *Graph illustrating the number of running compute nodes the first 11 days of long term test, and the same graph which combines original state and prototype state*

6 the power consumption of compute05 is displayed to see what happens when a physical server receives several virtual machines. The power consumption at compute05 spikes the minutes it is receiving virtual machines. The server consumed 100 watt more when receiving virtual machines but after a couple of minutes it goes back to normal again. The graph to the right illustrates the total consumption the environment while the prototype executed for the first time. It clearly shows a significant decrease of the consumption when 5 compute nodes where shutdown.

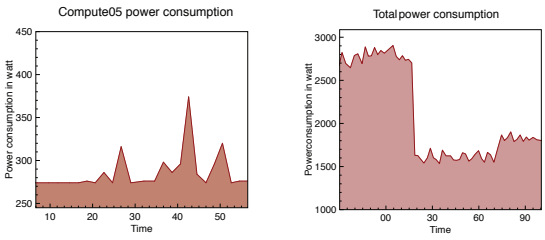

Figure 6. *Graph illustrating the power consumption of compute05 when it receives virtual machines and the total consumption of the environment*

The two graphs 7 illustrates the number of running compute nodes for the first 11 days of the prototype. The policy was set to have 2 machines as padding during the day to handle new bursts of virtual machines and 1 during night. From 06:00 to 18:00 there where 2 in padding while it from 18:00-06:00 was one extra machines. The graph to the right also have a grey field which indicated the original state of the system if the policy was not running.

After the first execution of the prototype the rest of the executions are left doing only little adjustment to the environment. This is also the reason for choosing policy 2 and not policy 3. Policy 3 would constantly moving virtual machines for every run. The lines in the next graph 8 is the number of virtual machines located at each compute node. As one can see there is not much movements in the environment only smaller adjustments after different virtual machines have been created.

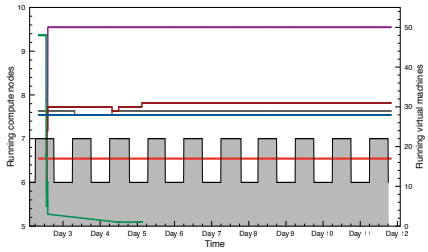

Figure 8. *Graph illustrating the number of running nodes with Y axis to the left and the number of virtual machines during long term test with Y axis to the left*

To illustrate how small the adjustments the prototype does after the first run, the graph 9 displays the live migrations done by the prototype.

To take a closer look at the actual power consumption and what the policy saved the number is presented in the table below.

Power consumption:	Policy 2 long term	No policy
Watt usage 1 day	43 992	74 448
3 weeks usage	923 kwh	1563 kwh
1 year estimated usage	16 057 kwh	27 173 kwh
Power consumption reduction	40%	0%

Table II
Long term test of policy 2. Three weeks with policy 2, 2 spare machines at daytime and 1 in the night. No performance constraints.

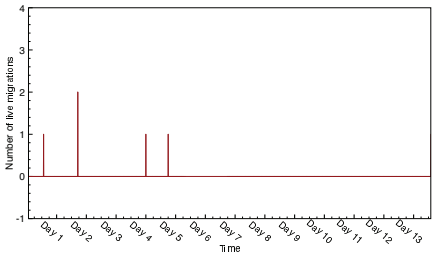

Figure 9. *Graph illustrating the number of live migrations during the long term test*

V. Conclusion and future work

The purpose of this study was to enable a cloud environment to scale dynamically to the needs of its users, whilst saving energy and heat. Our findings show that even simple bin packing algorithms are able to provide a useful mechanism for reducing the number of needed compute nodes. In the future we will attempt to incorporate workload optimization algorithms and to enable the cloud to switch between different strategies. We also want to investigate other types of constraints, such as disallowing some pairs of virtual machines to co-exist in the same compute node for high-availability and separation of IO loads.

References

[1] Dhiman, G., Marchetti, G., and Rosing, T. vgreen: A system for energy-efficient management of virtual machines. *ACM Trans. Des. Autom. Electron. Syst. 16*, 1 (Nov. 2010), 6:1–6:27.

[2] Hermenier, F., Lawall, J., and Muller, G. Btrplace: A flexible consolidation manager for highly available applications. *Dependable and Secure Computing, IEEE Transactions on 10*, 5 (Sept 2013), 273–286.

[3] Martello, S., Pisinger, D., and Vigo, D. The three-dimensional bin packing problem. *Operations Research 48*, 2 (2000), 256–267.

[4] Martello, S., Pisinger, D., Vigo, D., Boef, E. D., and Korst, J. Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem. *ACM Trans. Math. Softw. 33*, 1 (Mar. 2007).

[5] Mishra, M., and Sahoo, A. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on* (2011), IEEE, pp. 275–282.

[6] Nathuji, R., and Schwan, K. Virtualpower: Coordinated power management in virtualized enterprise systems. *SIGOPS Oper. Syst. Rev. 41*, 6 (Oct. 2007), 265–278.

[7] von Laszewski, G., Wang, L., Younge, A., and He, X. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on* (Aug 2009), pp. 1–10.

[8] Yanagisawa, H., Osogami, T., and Raymond, R. Dependable virtual machine allocation. In *INFOCOM, 2013 Proceedings IEEE* (April 2013), pp. 629–637.

[9] Zhang, Q., Zhani, M. F., Zhang, S., Zhu, Q., Boutaba, R., and Hellerstein, J. L. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proceedings of the 9th international conference on Autonomic computing* (2012), ACM, pp. 145–154.

[10] Zhang, Z., Guo, S., Zhu, W., Oon, W.-C., and Lim, A. Space defragmentation heuristic for 2d and 3d bin packing problems. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence* (2011), AAAI Press, pp. 699–704.