Release-Time Aware VM Placement

Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani† and Ammar Rayes‡
Oregon State University, Corvallis, OR 97331, dabbaghm,hamdaoub@onid.orst.edu
† Qatar University, mguizani@ieee.org
‡ Cisco Systems, San Jose, CA 95134, ‡ rayes@cisco.com

***Abstract*—Consolidating virtual machines (VMs) on as few physical machines (PMs) as possible so as to turn into sleep as many PMs as possible can make significant energy savings in cloud centers. Although traditional online bin packing heuristics, such as Best Fit (BF), have been used to reduce the number of active PMs, they share one common limitation; they do not account for VM release times, which can lead to an inefficient usage of energy resources. In this paper, we propose several extensions to the original BF heuristic by accounting for VMs' release times when making VM placement decisions. Our comparative studies conducted on Google traces show that, when compared to existing heuristics, the proposed heuristic reduces energy consumption and enhances utilization of cloud servers.**

***Index Terms*—Resource allocation, cloud computing, energy efficiency, VM Placement, Google traces.**

I. Introduction

Energy efficiency is a problem of a primal concern in cloud centers. Cloud providers are seeking every possible opportunity to save energy as the electricity bills associated with running cloud centers are in the order of billions of dollars per year [1]. There are also increasing environmental concerns about reducing energy in large data centers after reporting that the information technology is responsible for 2% of the global carbon emissions [2]. These financial and environmental concerns motivated researchers to find efficient ways to save energy in cloud centers.

A cloud center is made up of a huge number of servers, often referred to as physical machines (PMs). Cloud clients may, at any time, submit a request to the cloud cluster specifying the amount of resources they need. Upon receiving a request, the cloud scheduler creates a virtual machine (VM) for the requested resources and assigns it to a PM. The VM is reserved for a certain period, called the execution time, after which it is released. The virtualization technology allows scheduling multiple VMs on the same PM. This multiplexing capability brings a very effective way to save energy called VM consolidation which is based on the idea of packing the received VMs in as few active PMs as possible, thereby saving energy by putting to sleep as many redundant PMs as possible.

Previous work [3, 4] treated the problem of how to make efficient VM consolidation decisions as an online Bin Packing optimization problem, which views VMs as objects and PMs as bins and where the objective is to pack these objects in as few bins as possible. The problem is known to be NP-hard [3], and thus approximation heuristics were proposed [5] instead. Experiments showed that the Best Fit (BF) heuristic is one of the most effective approximation heuristics for this problem [6]. Its main limitation, however, is that it does not account for VMs' execution times (and thus for their release times) when making placement decisions, leading to inefficient cloud resource usage.

Fig. 1 (a) shows an example of such an inefficient placement made by the BF heuristic for four VMs where the indices reflect the chronological order of the submission of these VMs (i.e. V_1 was submitted before V_2 and so on). The amount of requested resources for V_1, V_2, V_3 and V_4 is: 55%, 50%, 40% and 45%, respectively, and the time after which these VMs will be released is: 3, 15, 10 and 4 hours, respectively. Both of the PMs (P_1 and P_2) each has a unit capacity. Observe that although V_1 and V_4 will be released shortly, the two PMs P_1 and P_2 will still be kept ON to accommodate the remaining VMs. In fact, they can be turned to sleep only after their last hosted VM is released. The placement shown in Fig. 1 (a) is clearly not energy efficient over time as P_1 and P_2 will be left ON for long periods with low utilization.

Researchers were alerted to the fact that termination of VMs at varying times will lead to a large number of ON PMs with sparse workload. Previous work [7] addressed this by performing periodic VM-PM remapping where some VMs are migrated to be consolidated on few PMs, allowing the rest to be turned to sleep. The main problem with this approach is that VM migration does not come for free in terms of performance degradation and energy overhead [8].

Rather than using remapping techniques to address the inefficient initial placements made by the traditional BF heuristic, we propose enhancing this heuristic by considering also the execution time of VMs (and thus their release times) when making placement decisions. Our enhanced heuristic produces a placement such as that in Fig. 1 (b) where VMs with short execution times are assigned to P_1, allowing it to turn to sleep after 4 hours and thus leaving a single PM ON with high utilization.

Very few works took the release time of VMs into account when making placement decisions. The authors in [9, 10] suggested a release-time aware heuristic that places a submitted VM V_i on a PM P_j only if the difference in the release time between V_i and all the VMs hosted on P_j is less than a certain threshold. If multiple ON PMs meet this condition, then the one with the least slack is selected. A new PM is switched from sleep to accommodate V_i if no

already ON PM satisfies this condition. The main problem with this approach is that its performance is highly sensitive to the selected threshold and there is no automatic way to find an appropriate threshold. The main advantage of our heuristic over prior release-time aware heuristics [9, 10] is that it is parameter free. Throughout the paper, we refer to the heuristic proposed in [9, 10] by *threshold-based heuristic*.

To sum everything up, the main contribution of our work is in proposing extensions to the BF heuristic that consider also the release time of requests in order to produce more energy efficient placements over time. Our heuristic is parameter free which makes it more practical to use compared to threshold-based techniques whose performance is highly dependent on the selected threshold.

Comparative studies conducted on real Google traces show the effectiveness of our heuristic compared to the original BF heuristic and to the prior release-time aware heuristics [9, 10] in terms of reducing the number of active PMs and saving energy. The assumption in this paper is that the time when the VM will be released is know at the submission time. This information can be directly specified by the client or can be predicted based on the nature of submitted VM task as in [11] or based on client's historical behavior.

The remainder of the paper is organized as follows. Section II describes our extensions for the BF heuristic. Section III presents our evaluation results conducted on Google traces. Section IV concludes the work and provides directions for future work.

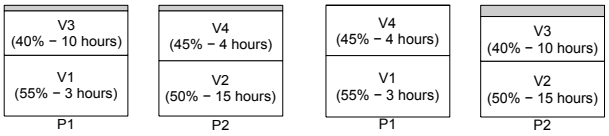

Fig. 1: Placement decisions for a) BF heuristic b) proposed release-time aware BF heuristic.

II. Release-Time Aware BF Heuristic

Let P be the set of all PMs in the cloud cluster where each PM $P_j \in P$ has a certain CPU capacity C_j. Suppose that client i submits at time s_i a VM request in the form of (w_i, d_i), where w_i is the amount of requested CPU resources and d_i is the duration for which these resources are needed. Since the submitted request needs to be scheduled immediately, then the release time of this request is: $r_i = s_i + d_i$. Let N_j denote the set of all VMs that are currently hosted on P_j. Then the aggregate used resources for any PM P_j can be calculated as: $U_j = \sum_{i \in N_j} w_i$.

The original BF heuristic attempts to reduce the number of active PMs by switching a new PM ON to host the submitted request only if the submitted request cannot be fitted in any of the already ON PMs. If multiple already ON PMs have enough slack to host the request, then the request is placed on the one with the least slack, where the slack for P_j is defined as $S_j = C_j - U_j$.

After describing how the BF heuristic works, it is clear that the execution times of VMs (and thus their release times) are completely ignored during the placement decisions. We extend the classical BF heuristic by taking into account VMs' release times in order to reduce the number of active PMs over time. Our extension consists basically of introducing and using a new metric (different from the slack) when making preference among multiple ON PMs that can fit the submitted request. Our proposed metric combines the following components:

a)- Temporal Slack: Rather than merely calculating the slack based on the current used resources, a natural extension of the BF heuristic would be to add a time dimension and to calculate the temporal slack throughout the whole execution time of the new submitted VM. This will capture the amount of left slack over time as some of the already hosted VMs might be released within the execution time of the newly submitted VM. Higher preference should be given to PMs with lower temporal slack.

The following example illustrates the intuition behind the temporal slack. Consider the case where there are only two ON PMs (P_1 and P_2) each with a unit capacity. Assume that P_1 already hosts V_1 and V_2 where the reserved resources for these VMs are 10% and 70%, respectively, and thus the current used resources for P_1 is $U_1 = 80\%$. Assume that P_2 already hosts V_3 and V_4 where the reserved resources for these VMs are 25% and 50%, respectively, and thus $U_2 = 75\%$. The time when all the hosted VMs will be released is also known. Now Suppose that the scheduler received at time s_5 a request V_5 to reserve $w_5 = 15\%$ and that this request will be released at r_5. The scheduler obviously has to choose between P_1 and P_2.

The original BF heuristic places the received request on P_1 as it has lower slack compared to P_2. The intuition here is that by this placement, the newly used resources will be $U_1 = 95\%$ and $U_2 = 75\%$ and thus we will be forced to switch ON a new PM from sleep in the future only if a request with $w > 25\%$ is received. Whereas if the request was placed on P_2, the newly used resources would be $U_1 = 80\%$ and $U_2 = 90\%$ and in that case any request with $w > 20\%$ would require turning a new PM ON. This shows the intuition behind the original BF heuristic as it places newly submitted requests on the PMs with the smallest slacks, leaving PMs with the largest slacks for supporting future VM requests with larger sizes.

However, we believe that the temporal slack would be a better metric than merely the resource slack when choosing PMs (e.g., choosing between P_1 and P_2), especially when the PM with the smallest slack (e.g., P_1) has VMs that will be released very soon. This can be explained by first showing the temporal slack shaded in gray for P_1 and P_2 in Fig. 2 where the vertical side represents the capacity of the considered PM whereas the horizontal side represents a time window that starts from s_5 and goes until r_5. The release time of the

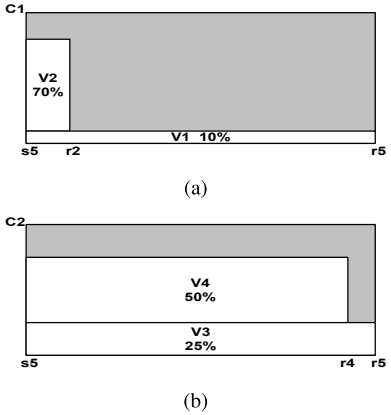

Fig. 2: The temporal slack shaded in gray for a) P_1 and b) P_2

VMs is also indicated on the horizontal side if it falls within s_5 and r_5. Observe that based on the indicated release times, P_2 has lower temporal slack than P_1 indicating that P_2 is a better place to host V_5. The intuition behind this preference is as follows. If V_5 is scheduled on P_1, then a new PM will be switched ON if we receive any request that has $w > 25$ in the short period between s_5 and r_2 or if we receive a request with $w > 75$ in the period between r_2 and r_5. Whereas if V_5 is scheduled on P_2, then a new PM will be switched ON if we receive a request with $w > 20$ in the short period between s_5 and r_2 or if a new request is submitted with $w > 90$ in the remaining long period between r_2 and r_5. So by placing V_5 on P_2, it is true that we cover smaller range of w at the beginning short period but larger ranges can be supported later for a longer period.

Formally, when considering a VM request V_i whose CPU demand is w_i and whose start, release, and execution times are s_i, r_i, and $d_i = r_i - s_i$, PM P_j's temporal slack α_{ij} can be calculated as:

$$\alpha_{ij} = C_j d_i - \sum_{k \in N_j} w_k (\min\{r_k, r_i\} - s_i)$$

Where the temporal slack α_{ij} is measured in (CPU×Time) unit.

b)- Uptime Extension: The uptime of PM P_j, termed τ_j, represents the time after which P_j becomes empty and hence can be turned to sleep. That is, τ_j is equal to the largest release time of the VMs hosted on P_j ($\tau_j = \max\{r_k \,|\, k \in N_j\}$). When selecting a PM for placing a VM request V_i with a release time r_i, PMs whose $\tau < r_i$ should be avoided, as doing so extends the PM's uptime, thus prolonging the duration after which the PM can be turned to sleep. If the submitted request can only fit into one of these PMs, then the one whose uptime will be extended the least should be selected. Based on this discussion, we now introduce and define the uptime extension metric of PM P_j when considering placing a VM request V_i with a release time r_i as $\beta_{ij} = C_j \times \max\{0, r_i - \tau_j\}$. The reason for including the PM's capacity C_j in this term is to make the uptime extension metric β_{ij} measured in (CPU×Time) unit similar to the temporal slack metric. Now when selecting a PM for placing the request V_i, PMs with lower uptime extension metric should be preferred over the rest of the PMs.

Now putting it all together, we introduce and use a combined metric, referred to as γ_{ij}, that combines both the temporal slack metric, α_{ij}, and the uptime extension metric, β_{ij}. The combined metric γ_{ij} of PM P_j when considering placing a VM request V_i is defined as $\gamma_{ij} = \alpha_{ij} + \beta_{ij}$. The PM P_j with the lowest γ_{ij} among those that can fit the submitted VM V_i is thus selected for placement.

Complexity Analysis: our enhanced heuristic has a computation complexity similar to that of BF where the ON PMs are first divided into those that can fit the request and those that can't and The PMs metrics are later compared in order to find the PM with the least metric. The only difference is in the calculated metric as for each ON PM that can fit the request, our heuristic calculates the temporal slack and the uptime extension rather than merely calculating the current resource slack as in the BF heuristic. However, this can be calculated directly by plugging the release times of the VMs and the capacity of the PM in the previously presented equations. There is also a storing overhead as our enhanced heuristic requires storing the release time of all the currently hosted VMs. However, this extra overhead is very small for large cloud clusters and leads into great energy savings and utilization gains as we will be seen in the following section.

III. Comparative Evaluation

The comparative evaluations presented in this section are based on real Google cluster data [12] that was released in November 2011 and consists of a 29-day traces collected from a cluster that contains more than 12K PMs. Three types of PMs are distinguished in the cluster in terms of the supported CPU capacity. Table I shows the number of PMs for each one of these types along with their CPU capacities normalized with respect to the PM with the largest capacity in the cluster. Since the size of the Google traces is huge, we only limit our analysis to a chunk of the traces that spans 30 hours. We evaluate how efficient the placements made by our heuristic are when considering all requests submitted within this period. For each VM request, the traces reveal the submission time, the release time and the requested CPU resources reported as a percent of maximum supported capacity in the cluster.

TABLE I: Configurations of the PMs within the Google cluster

Number of PMs	CPU Capacity
798	1.00
11658	0.50
126	0.25

We compare the placements made by our heuristic with those made by the following 3 heuristics: Any Fit (which

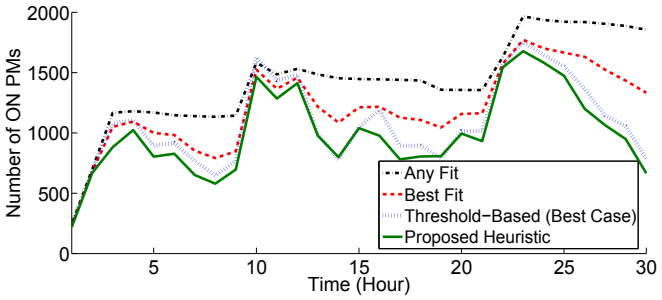

Fig. 3: Number of ON PMs over time when different heuristics are used to place the requests submitted to the Google cluster.

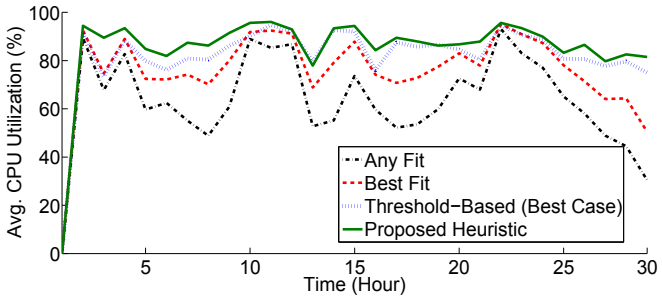

Fig. 4: Average Utilization over time when different heuristics are used to place the requests submitted to the Google cluster.

picks an ON PM at random among those that can fit the request), Best Fit, and the threshold-based heuristic proposed in [9, 10]. Since the performance of the threshold-based heuristic is dependent on the selected threshold and since there is no automatic way to tune that threshold, we evaluate this heuristic on the whole 30-hour traces using different threshold values. We report in our evaluation the results of the threshold that used the least number of PMs during the whole 30 hours. This represents the best-case scenario and so we refer to it by the *threshold-based heuristic (best case)*. It is worth mentioning that during our evaluation, we observed that the performance of this heuristic exhibits high variation/sensitivity to threshold values, which is one its key limitation.

For all of the evaluated placement heuristics, when the request can't be fitted in any already ON PMs, the PM in the sleep state with the largest capacity is switched ON to accommodate the received request. The intuition behind selecting the PM with the largest capacity is that it allows to fit a larger number of requests in the future. In our evaluation, we adapt a power management scheme that turns a PM to sleep once it becomes idle. More advanced predictive power management schemes (e.g. [13]) can be also combined with our placement heuristic where idle PMs that will be needed in the near future are kept ON to avoid the switching overhead, making greater energy savings.

We present next our heuristic comparative analysis in terms of the number of active PMs, utilization gains, and energy savings.

A. Number of Active PMs

We analyze first the number of PMs that were left ON to accommodate the received requests since it has a direct impact on the amount of consumed energy. Fig. 3 shows this number based on the submitted VM requests reported in the 30-hour period of the Google traces when different heuristics were used for placement. Observe that Any Fit heuristic had the worst results as it used the largest number of PMs to accommodate the received VM requests. This shows that randomly selecting one of the ON PMs to host the submitted request leads to inefficient VM placements. Observe that by selecting the ON PM with the least slack, the Best Fit heuristic uses lesser number of PMs when compared to Any Fit. However, the threshold-based heuristic under a well-tuned threshold uses lesser number of PMs when compared to Best Fit and Any Fit as it takes the release time of VMs into account when making new placements. Observe that our proposed heuristic used the least number of PMs all the time when compared to all other heuristics (including the threshold-based heuristic in its best case), as it takes the release time into account in its placement decisions by using a metric that combines both temporal slack and uptime extension.

B. Utilization Gains

We analyze next the utilization gains that can be achieved by our heuristic when compared to the other placement heuristics. The utilization of a PM P_j referred to as ν_j can be calculated as $\nu_j = U_j/C_j$, where U_j is the aggregate reserved resources for all VMs hosted on P_j and C_j is P_j's CPU capacity.

Let P_{on} denote the set of ON PMs in the cluster. Then the average CPU utilization ν_{avg} can be calculated as:

$$\nu_{avg} = \frac{\sum_{j \in P_{on}} \nu_j}{|P_{on}|}$$

where $|P_{on}|$ represents the number of ON PMs.

Fig. 4 shows the average CPU utilization ν_{avg} for the different heuristics over time. Observe that our heuristic achieves the highest average utilization among all the other heuristics at all time. This shows that our heuristic packs the submitted requests tightly, resulting in increasing the utilization of the ON PMs, and thus, in reducing the need for switching new PMs ON.

C. Energy Savings

We assess next the energy savings that our heuristic achieves by relying on the energy costs reported in [14] in our energy evaluations. As for the power consumption of ON PMs, the power increases linearly as the PM's CPU utilization level increases. The power of an ON PM, P_j^{on}, as a function of ν_j can be calculated as $P_j^{on}(\nu_j) = P^{idle} + \nu_j(P^{peak} - P^{idle})$ where P^{idle} is the consumed power under

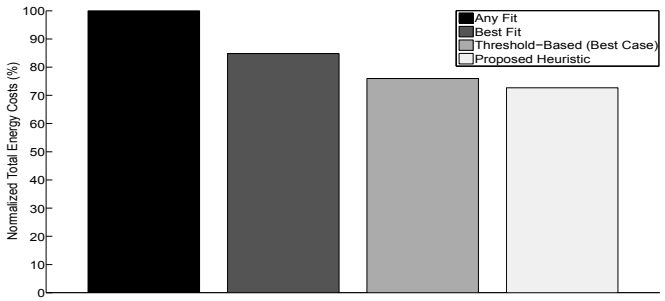

Fig. 5: Total Energy Costs for running the Google cluster under different placement heuristics (normalized w.r.t. Any Fit costs).

zero utilization, which is set to be equal to 200 Watts. P^{peak} is the consumed power at the peak load and is set to be equal to 400 Watts. A sleeping PM, on the other hand, consumes about $P^{sleep} = 100$ Watts. The transition energy consumed when switching a PM from sleep to ON is $E_{s \to o} = 4260$ Joules, and that when switching a PM from ON to sleep is $E_{o \to s} = 5510$ Joules. These energy/power numbers are taken from [14].

We calculate the total energy to run the cluster under the different placement heuristics where the total energy is the summation of the energy consumed by both ON and sleep PMs in addition to the switching energy (from sleep to ON and vice versa). Fig. 5 shows these total costs throughout the whole 30-hour period normalized with respect to the total energy cost of Any Fit heuristic. Observe that the release-time aware heuristics (our approach and the threshold-based heuristic) achieve great energy savings when compared to the classical Bin packing heuristics (Any Fit and Best Fit). Observe that our heuristic achieves the highest energy savings when compared to all other heuristics, including the Threshold-Based heuristic in its best case scenario. It is worth mentioning that when trying the threshold-based heuristic using different threshold values, some of these threshold values achieve no energy savings when compared to the Best Fit and some even achieve worse results than the Best Fit. We also show in our comparative studies that our heuristic achieves better results when compared to the threshold-based heuristic in its best case.

IV. Conclusion and Future Work

We have extended the Best Fit heuristic to consider also the release time of VMs when making placement decisions. Comparative studies conducted on real Google cluster traces show that our proposed heuristic reduces significantly the number of active PMs, resulting in great energy savings and utilization gains when compared to previously proposed placement heuristics. For future work, we plan to generalize the proposed heuristic to handle requests with multiple resources such as CPU, memory and bandwidth. Another interesting direction would be to predict release times of submitted requests based on historical traces in the case when VMs' release times are not specified by the client.

V. Acknowledgment

This work was supported in part by Cisco (CG-573228) and National Science Foundation (CAREER award CNS-0846044).

References

[1] R. Brown et al., "Report to congress on server and data center energy efficiency: Public law 109-431," 2008.

[2] C. Pettey, "Gartner estimates ICT industry accounts for 2 percent of global co2 emissions," 2007.

[3] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," *Microsoft Technical Report*, 2011.

[4] A. Beloglazov, *Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing*, Ph.D. thesis, 2013.

[5] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation," in *Proc. of the Computer Measurement Group Conf.*, 2007.

[6] Bo L., Jianxin L., Jinpeng H., Tianyu W., Qin L., and Liang Z., "Enacloud: An energy-saving application live placement approach for cloud computing environments," in *IEEE Int'l Conference on Cloud Computing*, 2009.

[7] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2010.

[8] H. Liu, H. Jin, C. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster computing*, vol. 16, no. 2, pp. 249–264, 2013.

[9] M. Cardosa, A. Singh, H. Pucha, and A. Chandra, "Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1737–1751, 2012.

[10] Y. Li, Y. Wang, B. Yin, and L. Guan, "An energy efficient resource management method in virtualized cloud environment," in *Network Operations and Management Symposium (APNOMS)*, 2012, pp. 1–8.

[11] M. Cardosa, P. Narang, A. Chandra, H. Pucha, and A. Singh, "Steamengine: Driving mapreduce provisioning in the cloud," in *International Conference on High Performance Computing (HiPC)*, 2011.

[12] *http://code.google.com/p/googleclusterdata/*.

[13] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient cloud resource management," in *INFOCOM Workshop on Moblie Cloud Computing*, 2014.

[14] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer Journal*, vol. 40, pp. 33–37, 2007.