Supplementary File: Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment

Zhen Xiao, *Senior Member, IEEE,* Weijia Song, and Qi Chen,

1 Analysis of Skewness Algorithm

The skewness algorithm consists of three parts: load prediction, hot spot mitigation, and green computing. Let n and m be the number of PMs and VMs in the system, respectively. The number of resources (CPU, memory, I/O, etc.) that need to be considered is usually a small constant (e.g., 3 or 4). Thus the computation of the skewness and the temperature metrics for a single server takes a constant amount of time. During load prediction, we need to apply the FUSD algorithm to each VM and each PM. The time complexity is O(n+m).

1.1 Complexity of hot spot mitigation

For hot spot mitigation, let n_h be the number of hot spots in the system during a decision. Sorting them based on their temperature takes O($n_h * \log(n_h)$). For each hot spot, we need to sort the set of VMs running on it. In practice, the number of VMs that run on a PM is typically limited to a small constant. Hence, the sorting takes a constant amount of time. For each VM, we need to scan the rest of the PMs to find a suitable destination for it, which takes O(n). The overall complexity of this phase is thus O($n_h * n$). Under normal condition, the system is likely to have only a few hot spots and the decision time is about linear with the system size.

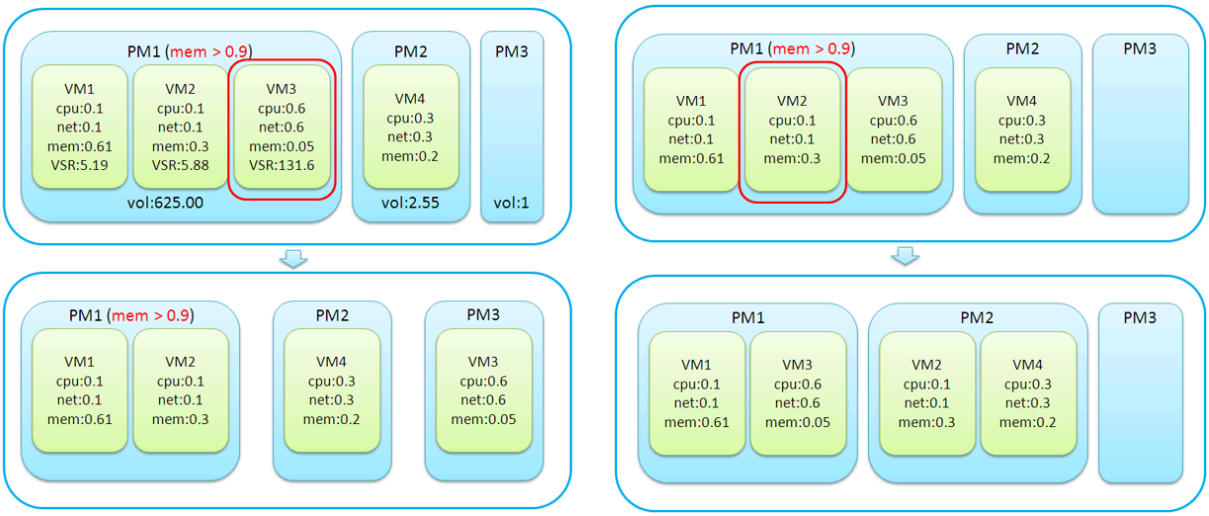
Readers familiar with the algorithm in [1] (we call it the BG algorithm) will notice that the complexity of our algorithm is higher than theirs. This is mostly due to the comprehensive search in finding a suitable destination server for a VM that needs to be moved away. In contrast, the BG algorithm combines multi-dimensional load information into a single *Volume* metric [1]. It sorts the list of PMs based on their volumes and the VMs in each PM in their volume-to-size ratio (VSR). It then considers the PMs and the VMs in the pre-sorted order when making the migration decision. This is faster than our algorithm because their search space is smaller, but it may not work correctly under some scenarios.

• *Z. Xiao, W. Song, and Q. Chen are with the Department of Computer Science, Peking University.*

Figure 1 shows an example where we have three homogeneous PMs in the system. We consider three types of resources: CPU, memory, and network. The first PM is overloaded in the memory dimension – the total memory consumed by its three VMs exceeds 90%. Clearly, one of the first two memory intensive VMs should be migrated away. However, VM_3 has the largest VSR due to its high load in other types of resources and its small memory footprint. As a result, their algorithm will migrate away VM_3 because doing so reduces PM_1's volume the greatest per unit of bytes moved. Unfortunately, this does little to mitigate the memory pressure on PM_1 which remains overloaded after the migration while its CPU and other resources are left mostly idle. (Like ours, their algorithm offloads at most one VM for each hot spot during a decision run.) What it fails to realize is that the high volume of VM_3 is not contributed by the overloaded resource and that the large VSR of VM_3 is precisely because it contributes little to the overloaded resource. In contrast, the figure shows that our algorithm will migrate away VM_2 because its removal reduces the temperature and the skewness of PM_1 the greatest. This eliminates the hot spot successfully. Another difference illustrated in the figure is the selection of the destination server. The BG algorithm selects a destination server in the increasing order of volume (i.e., it favors low volume servers first). Because an idle server inevitably has the least volume, their algorithm tends to spread the VMs onto idle servers during offloading. (Unlike our algorithm, theirs does not have a subsequent green computing phase and hence those VMs are never consolidated when the load goes down.) In this example, it will migrate VM_3 to PM_3. In contrast, our algorithm will migrate VM_2 to PM_2 because doing so reduces the skewness of the destination PM the greatest. (Migrating VM_2 to PM_3 will actually increase the skewness of PM_3.) The load of PM_2 remains relatively low ($\leq$ 50% for all resources) after the migration. Hence, it is a better destination from an energy conserving point of view.

1.2 Complexity of Green Computing

For the green computing phase, let n_c be the number of cold spots in the system during a decision run. Sorting them based on their memory sizes takes O($n_c * \log(n_c)$). For each VM in a



(a) blackbox algorithm (b) skewness algorithm

Fig. 1. Comparison of hot spot mitigation between two algorithms

cold spot, it takes O(n) time to find a destination server for it. Recall that we assume the number of VMs on a PM is no more than a small constant. Hence, the overall complexity of this phase is $O(n_c * n)$. It seems that this can get as high as $O(n^2)$ when the number of cold spots is large. However, we restrict the number of cold spots that can be eliminated in each run of the algorithm to be no more than a certain percentage of active servers in the system. As we will see later in the simulation, when this limit is small, the running time is mostly linear with the system size.

It is easy to see that the overall complexity of the algorithm is bounded by $O(n^2)$. The hot spot mitigation phase and the green computing phase are typically not invoked together during the same decision run.

1.3 Number of migrations

During hot spot mitigation, each hot spot incurs at most one migration. During green computing, the number of migrations is bounded by the number of VMs on the cold spots. The total number of migrations is bounded by O(n).

2 Impact of Thresholds on the Number of APMs at Extreme Situation

To examine the performance of our algorithm in more extreme situations, we also create a synthetic workload shown in the right figure. The load mimics the shape of a sine function (only the positive part) and ranges from 15% to 95% with a 20% random fluctuation. It has a much larger peak-to-mean ratio than the real trace. We use the default parameters for our algorithm. Figure 2 shows that our algorithm adapts to this workload even better than real trace, taking and releasing PMs as needed.

3 Load Balance

We use the Gini coefficient [2] to measure the distribution of resource utilizations across APMs. Gini coefficient is widely

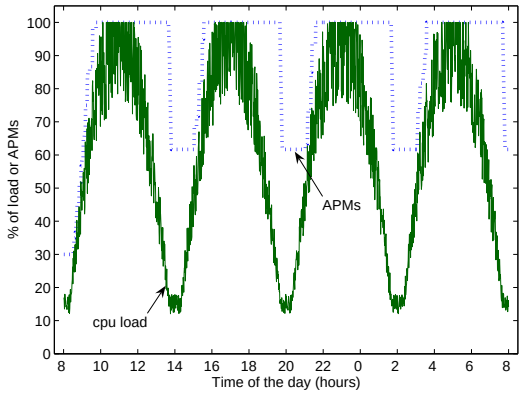


Fig. 2. Impact of thresholds on the number of APMs

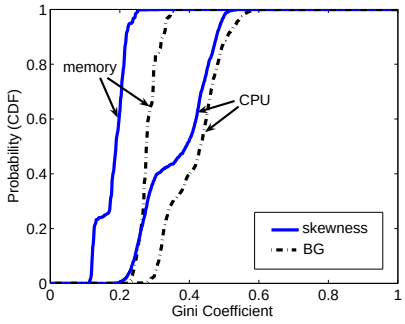


Fig. 3. Gini index of CPU and memory

used to measure the distribution of wealth in society. It is calculated from the normalized area between the observed CDF and CDF of a uniform distribution. A small coefficient indicates a more equal distribution. Figure 3 (a) shows the CDF of Gini coefficient for the CPU and the memory resources. We compute the Gini coefficient for APMs only. As can be seen from the figure, our algorithm achieves a more balanced load distribution than the BG algorithm. Later

in section 4 we will see that the Gini coefficients in our algorithm get better with a larger VM to PM ratio.

4 VARYING VM TO PM RATIO

To evaluate the performance of our algorithm with different VM to PM ratios, we increase the number of VMs in the experiment from 25 to 750 by adding 25 VMs every 50 decision intervals. The set of VMs are chosen randomly from the trace. The number of PMs is fixed at 100. The results are shown in Figure 4. The left figure shows that the Gini coefficients for all resources decrease substantially as more VMs are added into the system. ('nettx' and ''netrx' denote network sends and receives, respectively.) With a higher VM to PM ratio, each VM contributes a smaller portion of the load to its PM. This gives our algorithm a finer granularity in adjusting the resource allocation. The surge at the beginning of the curves is due to the random VM to PM mapping in the initial layout. Later when the VMs are consolidated onto a smaller number of PMs by green computing, the Gini coefficients increases. However, as more VMs are added into the system, the load among the PMs becomes very well balanced. We also observe that the Gini coefficient for CPU exhibits a much larger variation than that for other resources. This is because the CPU load in our trace fluctuates with time the most significantly .

The rest of Figure 4 shows the average decision time and the number of migrations during each run of the algorithm under different load conditions. We use the CPU load as an example due to space constraints. Generally, a higher VM to PM ratio corresponds to a higher load in the system. For a given configuration, the load of PMs also varies with time according to the trace. We separate hot spot mitigation and green computing into sub-figures with different scales on the y-axis. We first look at hot spot mitigation (top two charts in Figure 4 (b) and (c)). We can see that the decision time generally increases with the system load. (There is inevitably some variation due to the load of other resources not shown here.) This is expected: When the load is high, more hot spots exist in the system and fewer destination PMs to offload them. The number of migrations also increases with the load until the system is heavily overloaded. In this case, it is no longer possible to eliminate hot spots (since almost all PMs are hot spots themselves). Our algorithm will artificially inflate the capacity of PMs and try one more time. It will not initiate fruitless migrations, however. This increases the decision time further, but the number of migrations actually decreases significantly (because the algorithm cannot find suitable destination PMs).

For green computing (bottom two charts in Figure 4 (b) and (c)), we can see that green computing happens mostly when the system is lightly loaded. The lower the load, the more the number of cold spots. However, lower load does not necessarily correspond to longer decision time – the decision time actually decreases when the load is really low. This is because we limit the number of cold spots that can be eliminated in each run of the algorithm to a small percentage (5%) of APMs. In addition, under such low load it is easy to consolidate the PMs without exceeding the warm threshold. Hence, the decision is faster. The right (bottom) figure also demonstrates the effect of the consolidation limit: the number of migrations increases as the load decreases but remains bounded when the load is really low.

TABLE 1
Resource utilization of VMs

VM	CPU	memory	network
1	0.1	0.31	0.1
2	0.56	0.05	0.46
3	0.1	0.63	0.1
4	0.31	0.2	0.37

5 COMPARISON WITH THE BLACK/GRAY BOX ALGORITHM

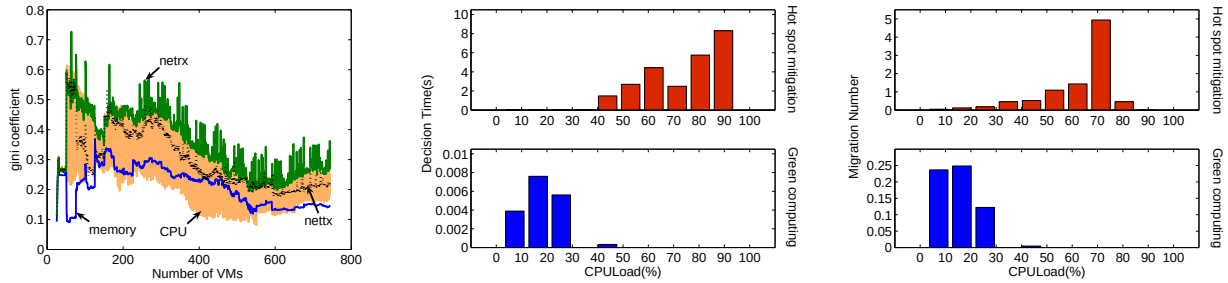
In Section 1, we give an example scenario which is handled correctly by our algorithm, but not by the BG algorithm. Here we explore their difference in real experiments. We use three PMs and four VMs as shown in Figure 5. Each row corresponds to one PM. The three columns represent CPU, memory, and network, respectively. Initially, VM_1, VM_2, and VM_3 are on PM_1 and VM_4 is on PM_2. Then we increase the load of the three VMs on PM_1 gradually to the level in table 1. This makes PM_1 a hot spot. The figure shows that the BG algorithm migrates VM_2 to PM_3, which unfortunately does not resolve the hot spot. Shortly after, it has to make another migration to move VM_1 to PM_2. In contrast, Figure 6 shows that the skewness algorithm needs only one migration to resolve the hot spot by moving VM_3 to PM_3. Another difference shown in the figures is green computing. After we decrease the load of all VMs to a low level, the skewness algorithm consolidates them onto PM_1 while the BG algorithm takes no action.

We want to emphasize that the BG algorithm differs from ours in many other aspects, such as their comparison of the blackbox vs. the graybox strategies, their load prediction and profile generation, etc.. Those differences are orthogonal to the discussion here. Using a lower hot threshold (e.g., 75% as in [1]) will not solve the problem, either.

6 COMPARISON WITH THE VECTORDOT [3] ALGORITHM

VectorDot is the scheduling algorithm used in HARMONY [3] virtualization system, whose architecture is similar to ours. It optimizes utilization for three types of resources including physical servers as well as data center network bandwidth and I/O bandwidth of the centralized storage system. Besides Virutal Machine migration, Virtual Storage migration is employed.

VectorDot introduced an index named extended vector product (EVP) to measure how much a virtual machine i contribute to the current resource utilization of its hosting physical server u. We use I to refer to the demand vector of i and U to the utilization vector of u. The value of $EVP(i, u)$



(a) Gini coefficients (b) decision time (c) number of migrations

Fig. 4. Varying VM to PM ratio

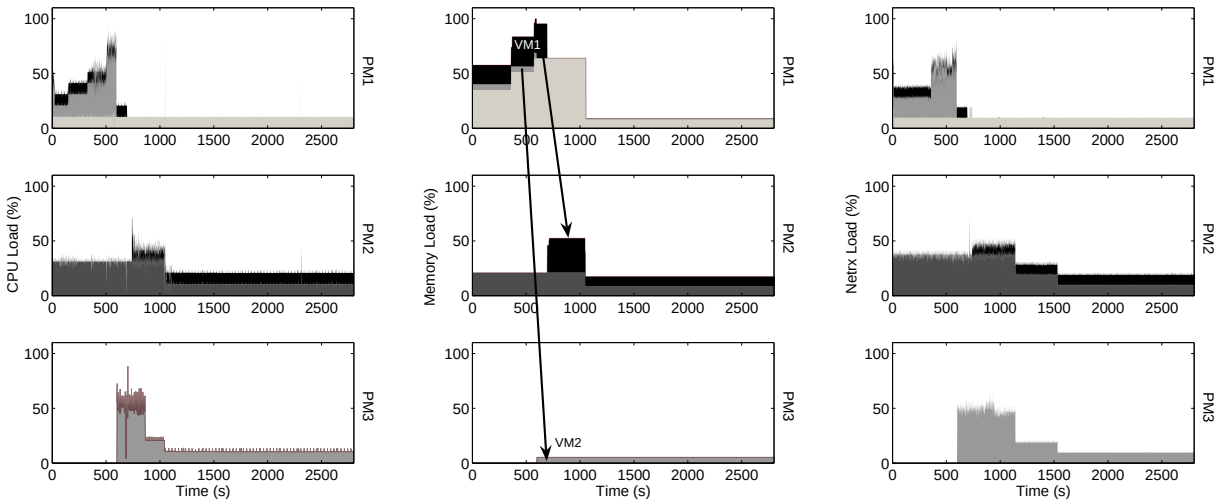


Fig. 5. Black/Gray box algorithm

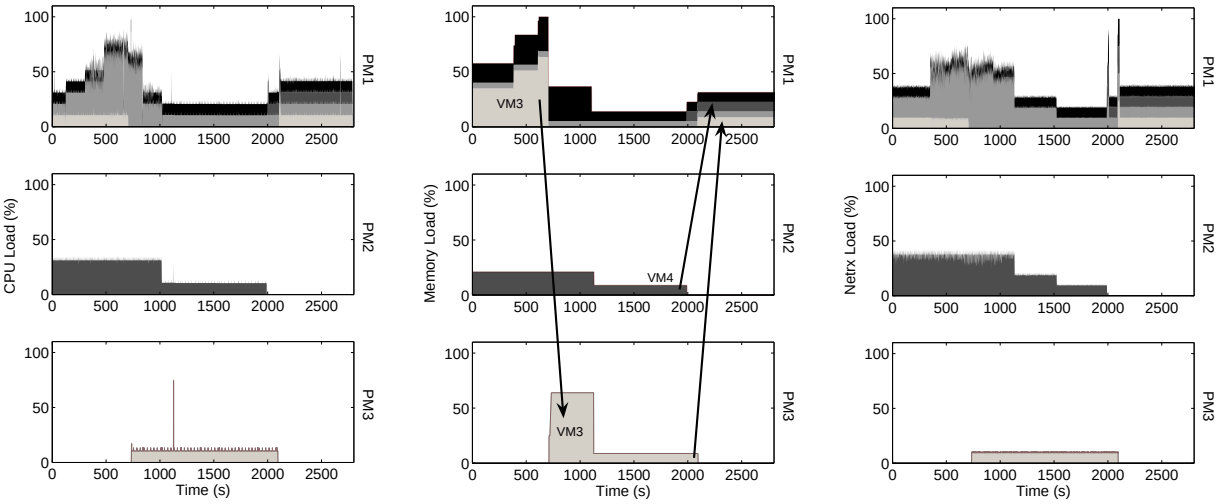


Fig. 6. Skewness algorithm

is determined by the dot product of I and U. Greater value of EVP means I contributes more to U. When more than one migration solutions available, VectorDot choose the one with minimal EVP. Our algorithm is similar to VectorDot by minimizing $Skewness$. However, VectorDot emphasizes on load balancing. Skewness explicitly takes care of the structure of residual resources to make them as useful as possible.

Suppose a virtual machine i is scheduled. There are two physical servers u_1 and u_2 which can accept i without breaking hot threshold. For simplicity, we consider only three resource dimensions in the following discussion. Give the resource demand vector of i is $I = (0.2, 0.05, 0.05)$.

The resource utilization vectors of u_1 and u_2 are $U_1 = (0.35, 0.5, 0.5)$ and $U_2 = (0.5, 0.05, 0.05)$ respectively. After accepting i, we can calculate $EVP(i, u_1) = 0.165$ and $EVP(i, u_2) = 0.15$. Therefore, VectorDot will migrate i to u_2. The residual resource vector of u_2 hence becomes $(0.3, 0.9, 0.9)$. The uneven distribution of residual resource makes it hard to be fully utilized in the future. On the contrary, Skewness is going to migrate i to u_1. The residual resource vector of u_1 hence becomes $(0.45, 0.45, 0.45)$, which is friendly to virtual machines.

Compared to VectorDot, in addition, our skewness algorithm supports load balance as well as green computing. We also adopt load predicting to improve scheduling effectiveness.

REFERENCES

[1] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. of the Fourth Symposium on Networked Systems Design and Implementation (NSDI'07)*, Apr. 2007.

[2] D. Ray, *Development Economics*. NJ: Princeton University Press, 1998.

[3] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *Proc. of the 2008 ACM/IEEE conference on Supercomputing*, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413424