

Sharing-Aware Online Algorithms for Virtual Machine Packing in Cloud Environments

Safraz Rampersaud
Department of Computer Science
Wayne State University
Detroit, Michigan USA
safraz@wayne.edu

Daniel Grosu
Department of Computer Science
Wayne State University
Detroit, Michigan USA
dgrosu@wayne.edu

Abstract—Cloud service providers offer services to a large number of users by employing virtualization technologies. One of the challenges faced by the cloud providers using virtualized environments is the development of efficient algorithms for assigning Virtual Machine (VM) instances to servers such that the number of hosting servers is minimized. This is also known as the problem of VM Packing. In this paper, we design a family of sharing-aware online algorithms for solving the VM Packing problem that take into account the sharing of memory among collocated VMs. We introduce a new server resource scarcity metric which establishes an order among servers that are suitable for hosting an arriving VM request and use it in the design of the algorithms. We evaluate the performance of our sharing-aware online algorithms by performing an extensive set of experiments comparing them against several existing sharing-oblivious VM packing algorithms.

Index Terms—online algorithms, sharing-aware VM packing, vector bin packing, virtual machines.

I. INTRODUCTION

Cloud adoption by government, industrial, and academic institutions have created opportunities for providers to offer services at scale through flexible infrastructures based on virtualization technologies. Industry forecasts predict that by 2018 approximately 80% of all workloads will be managed through data center virtualization services [1]. Current virtualization technologies incorporate mechanisms that perform *memory reclamation*, i.e., mechanisms that regulate/conservate memory resources when multiple VMs are instantiated through a hypervisor layer. The deduplication of similar memory pages between two or more VMs instantiated through the same hypervisor layer, i.e., *page-sharing*, is an example of such techniques. Fairly recently, frameworks supporting hypervisor-based VM page-sharing, such as Satori [2], Memory Buddies [3], and Difference Engine [4] have been developed. New processes and mechanisms, such as the aforementioned ones, drive the development of more efficient algorithms suitable for resource management. A variant of the VM resource allocation problem motivated by these developments in virtualization technology is the VM Packing problem [5].

The VM Packing problem considers instantiating multiple-type VMs in an offline setting which utilizes hypervisors as an architectural layer on top of physical servers, allowing for page-sharing; resulting in a reduced utilization of the memory

resource. Traditionally, VM allocation problems with multiple resource requirements have been modeled as vector bin packing problems [6], where each resource is represented as a vector component. The goal is then to minimize the number of active servers used in order to instantiate a set of VMs according to server allocation policies and available resource capacities. The classical sharing-oblivious vector bin packing algorithms in an online setting where VMs request multiple types of resources, will result in less efficient allocations since they do not leverage memory sharing opportunities. Therefore, in this paper, we design and investigate algorithms for solving the sharing-aware online VM Packing problem which results in a minimum number of active servers used to instantiate arriving VMs, where page-sharing occurs relative to VMs already instantiated on the servers. Since the hypervisors used by cloud providers employ memory reclamation techniques, our proposed sharing-aware algorithms leverage these techniques and are able to significantly reduce the number of servers needed to satisfy the users' requests and implicitly lead to a significant reduction of energy and costs for the cloud providers.

A. Our Contribution

We propose and design sharing-aware online algorithms for solving the VM Packing problem with multiple resource requirements, in an online setting. Our proposed sharing-aware online algorithms are improved designs of classical online algorithms for vector bin packing, that is, our designs take memory sharing into account when making allocation decisions. We introduce a new *server resource scarcity* metric necessary for designing sharing-aware Best-Fit and Worst-Fit type algorithms. Our *server resource scarcity* metric considers all VM resource requirements, server's available resource capacities and page-sharing. The metric is used to identify an active server with the highest priority to instantiate an online VM request. We perform extensive experiments to compare the performance of our sharing-aware online algorithms against several sharing-oblivious algorithms for VM packing. To the best of our knowledge, no sharing-aware online algorithms for packing VMs with multiple resource requirements have been proposed in the research literature to date.

B. Related Work

Several variants of online vector bin packing problem modeling the allocation of resources in clouds have been recently investigated. Song *et al.* [7] proposed a semi-online bin packing algorithm for resource allocation. Their proposed setup allows VMs to be reshuffled through live migration among the servers if resource conservation can be achieved. Li *et al.* [8] introduced a novel variant of the bin packing algorithm which attempts to minimize the total cost associated with a server's utilization. Kamali and Ortiz [9] improved upon the upper bound for Next-Fit and introduced a new algorithm, Move To Front, which performed the best in the average case for the online dynamic bin packing total cost minimization problem. Resource awareness is a prevalent topic in designing resource allocation algorithms for cloud environments. Carli *et al.* [10] formulated a variant of the bin packing problem, called Variable-Sized Bin Packing with Cost and Item Fragmentation, which is energy-aware when attempting to pack cloud resource requests onto servers in both online and offline settings. Breitgand and Epstein [11] considered a variant of the bin packing problem called Stochastic Bin Packing (SBP) which is risk-aware of network bandwidth consumption, and designed both online and approximation algorithms to solve it. Kleineweber *et al.* [12] proposed a variant of the multi-dimensional bin packing problem which is QoS-aware relative to cloud file systems, specific to storage virtualization. They designed an object-based file system reservation scheduler which manages the mapping of storage area volumes onto available object storage devices while observing QoS requirements. None of the research presented above considered memory sharing when making resource management decisions. Fairly recently, several systems such as Satori [2], Memory Buddies [3], and Difference Engine [4], considered hypervisor-based VM page-sharing, but did not address the design of online sharing-aware algorithms for VM packing.

The closest research to ours is by Sindelar *et al.* [5] who were the first to propose and analyze *offline sharing-aware algorithms* for the VM Maximization and VM Packing problems under hierarchical models which capture page sharing among instantiated VMs. However, they do not consider an online setting and did not propose sharing-aware online algorithms for the VM Packing problem. In our previous work [13], [14], we considered the design of sharing-aware offline algorithms for the VM Maximization problem under the general sharing model. The VM Maximization problem is different from the VM Packing problem we are considering in this paper. The objective of the VM Maximization problem is to allocate VM instances onto a set of servers such that the profit is maximized, while the objective of the VM Packing problem is to minimize the number of servers used to host user requested VM instances.

Recent research investigated vector-bin packing algorithms in the context of VM resource management. For the offline setting, Panigrahy *et al.* [6] studied heuristic variants of the First-Fit-Decreasing algorithm for VM allocation. Based on

large scale simulations, their proposed heuristics determined a minimum number of servers close to the optimal solution. For the online setting, Azar *et al.* [15] proposed vector-bin packing algorithms, analyzed their performance under various VM sequences, and established a new lower bound for the competitive ratio. While competitive analysis has been the standard for evaluating online algorithms, determining *relative worst order ratios* between online algorithms' performance serves as another means of evaluating them. Boyar and Favrholdt [16] introduced the relative worst order ratio metric for online bin packing algorithms and provided an extensive analysis of the algorithms. None of these works investigated online algorithms for VM Packing that take into account the sharing of memory among collocated VM instances. To the best of our knowledge we are the first to design and investigate such algorithms.

C. Organization

The rest of the paper is organized as follows. In Section II, we define the Sharing-Aware Online VM Packing Problem. In Section III, we present the design of our proposed online sharing-aware algorithms for VM packing. In Section IV, we compare the performance of our proposed algorithms against several sharing-oblivious algorithms through extensive experiments. In Section V, we summarize our results and present possible directions for future research.

II. SHARING-AWARE ONLINE VM PACKING PROBLEM

We now describe the Sharing-Aware Online Virtual Machine Packing (SA-OVMP) problem from the perspective of a cloud service provider.

We consider a cloud service provider that offers resources in the form of VM instances to cloud users. A VM instance requested by a user is denoted by V_j and is characterized by a tuple $\langle q_j^u, q_j^m, q_j^s \rangle$, where q_j^u is the number of CPUs, q_j^m is the amount of memory, and q_j^s is the amount of storage requested for V_j . For simplicity of presentation and the relevance to practical settings, we specifically consider only three main types of resources that characterize a VM instance: CPU, memory, and storage.

The cloud service provider has a set $\mathcal{S}$ of servers available for instantiating the VMs requested by the users. Each server $S_k \in \mathcal{S}$ is characterized by the amount of available resources of each type, represented as a tuple $\langle C_k^u, C_k^m, C_k^s \rangle$, where C_k^u is the number of available CPUs; C_k^m is the available memory capacity; and, C_k^s is the available storage capacity. We denote by $\mathcal{R}$ the subset of resource types composed of CPUs (type denoted by u) and storage (type denoted by s), that is, $\mathcal{R} = \{u, s\}$. The memory resource (type denoted by m) is not included in $\mathcal{R}$ since in the design of our algorithms we will treat the memory resource differently by considering memory sharing among the VMs collocated on the same server. We assume that the requests for resources from any single VM instance can be satisfied by any server $S_k \in \mathcal{S}$ that does not already host a VM, i.e., $q_j^u \leq C_k^u$, $q_j^m \leq C_k^m$, and $q_j^s \leq C_k^s$. When several VM instances are hosted on a server S_k , and they use a common subset of memory pages,

Table I: Notation.

Expression	Description
V_j	Virtual machine j .
$\mathcal{S}$	Set of available servers.
$\bar{\mathcal{S}}$	Set of inactive servers; $\bar{\mathcal{S}} \subset \mathcal{S}$
S_{i+1}	First available inactive server from $\bar{\mathcal{S}}$.
q_j^u	Requested number of CPUs by V_j .
q_j^m	Requested amount of memory by V_j .
q_j^s	Requested amount of storage by V_j .
C_k^u	CPU capacity of server S_k .
C_k^m	Memory capacity of server S_k .
C_k^s	Storage capacity of server S_k .
$\mathcal{R}$	Subset of server resource types u and s , $\mathcal{R} = \{u, s\}$.
E_j^k	S_k server resource scarcity metric value relative to V_j .
s_j^k	Shared memory pages (in GB) relative to S_k and V_j .

the total amount of memory allocated to those VM instances can be reduced through page-sharing. For example, when two Microsoft Windows 8 VM instances are collocated on the same server, they can share a significant amount of pages and the total allocated memory to those two VM instances can be reduced significantly compared to the case in which sharing is not considered. To determine the amount of sharing among VM instances, the cloud provider uses a *staging* server that computes the memory fingerprints [3] of the VM instance that is ready for allocation on the main cloud servers. The fingerprint of the VM instance is then used to determine the amount of memory sharing, denoted by s_j^k , among the currently considered VM instance, V_j , and the VM instances that are already hosted by server S_k . The notation used in the paper is presented in Table I.

The cloud provider is interested in satisfying as many requests from users as possible, while using the minimum possible number of servers. The requests for VM instances arrive one by one and the cloud provider needs to decide the allocation of the newly arrived VM request without knowing the information about future requests. Thus, this is an online setting and the cloud provider has to rely on online algorithms to allocate the VMs to servers. Our goal is to design such online algorithms for VM packing that take the sharing of memory into account when making allocation decisions. Before doing that, we formulate the problem of Sharing-Aware Online VM Packing (SA-OVMP) as follows.

SA-OVMP problem: We consider a cloud provider having a set of servers, $\mathcal{S}$, where each server $S_k \in \mathcal{S}$ is characterized by $\langle C_k^u, C_k^m, C_k^s \rangle$, and a sequence of VM requests $\{V_1, V_2, \dots, V_j, \dots\}$, arriving one by one, where each VM request V_j is characterized by $\langle q_j^u, q_j^m, q_j^s \rangle$. A VM request must be assigned to a server $S_k \in \mathcal{S}$ upon arrival, so that the following capacity constraints are satisfied:

$$C_k^m - q_j^m + s_j^k \geq 0 \quad (1)$$

and

$$C_k^r - q_j^r \geq 0, \quad \forall r \in \mathcal{R}, \quad (2)$$

where s_j^k is the amount of memory sharing among the currently considered VM instance, V_j , and the VM instances that are already hosted by server S_k . The *objective* is to minimize the total number of active servers.

Equation (1) is the memory capacity constraint, guaranteeing that the available memory capacity at server S_k is not exceeded. The available capacity $C_k^m - q_j^m$ is adjusted for the amount of sharing, s_j^k , between V_j and the instances already hosted by S_k . The constraints in Equation (2) guarantee that the capacities of the other types of resources of server S_k are not exceeded.

III. SHARING-AWARE ONLINE VM PACKING ALGORITHMS

In this section, we design sharing-aware online algorithms for solving the SA-OVMP problem. Before describing the algorithms we introduce few definitions and assumptions concerning the servers. The servers managed by the cloud provider are in one of the following two states: active and inactive. An *active server* is a server that is powered on and is currently considered for allocation by the algorithms. An *inactive server* is a server that is not powered on and is not currently considered for allocation by the algorithms. We denote by $\bar{\mathcal{S}}$ the set of inactive servers. When all the VMs hosted by a server are terminated the server becomes an inactive server and can be activated in the future. Initially, all servers are inactive servers, i.e., $\bar{\mathcal{S}} = \mathcal{S}$. All the sharing-aware algorithms presented in the paper assume that s_j^k , the amount of sharing among the currently arrived VM V_j and the VMs hosted by active server S_k was already determined through the memory fingerprinting on the staging servers as described in Section II. We now present our first sharing-aware algorithm.

A. Next-Fit-Sharing Algorithm

In order to describe the Next-Fit-Sharing, we need to introduce a third type of state for servers, called closed server. A *closed server* is already hosting VM instances and is not currently considered for allocation by the algorithm. The Next-Fit-Sharing algorithm is given in Algorithm 1 and works as follows. Only one server is active at the time. Upon arrival of

Algorithm 1 Next-Fit-Sharing

- 1: **Input:** VM instance arrival (V_j)
 - 2: $\{S_i$: currently active server. $\}$
 - 3: **if** $(C_i^m - q_j^m + s_j^i < 0)$ **or** $(C_i^u - q_j^u < 0)$ **or** $(C_i^s - q_j^s < 0)$ **then**
 - 4: Mark server S_i as closed
 - 5: $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \{S_{i+1}\}$
 - 6: Activate server S_{i+1}
 - 7: $S_{i+1} \leftarrow S_{i+1} \cup \{V_j\}$
 - 8: $i \leftarrow i + 1$
 - 9: **else**
 - 10: $S_i \leftarrow S_i \cup \{V_j\}$
 - 11: $[C_i^m, C_i^u, C_i^s] \leftarrow [C_i^m, C_i^u, C_i^s] - [q_j^m, q_j^u, q_j^s]$
-

V_j	S_1	S_2
$\langle q_j^u, q_j^m, q_j^s \rangle$	$\langle C_1^u, C_1^m, C_1^s \rangle, s_j^1$	$\langle C_2^u, C_2^m, C_2^s \rangle, s_j^2$
$V_1: \langle 4, 4, 2 \rangle$	$\langle 4, 6, 8 \rangle, 0$	$\langle *, *, * \rangle, *$
$V_2: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$
$V_3: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$
$V_4: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$

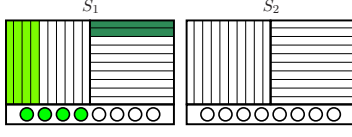


Figure 1: Next-Fit-Sharing: Allocating V_1

V_j	S_1	S_2
$\langle q_j^u, q_j^m, q_j^s \rangle$	$\langle C_1^u, C_1^m, C_1^s \rangle, s_j^1$	$\langle C_2^u, C_2^m, C_2^s \rangle, s_j^2$
$V_1: \langle 4, 4, 2 \rangle$	$\langle 4, 6, 8 \rangle, 0$	$\langle *, *, * \rangle, *$
$V_2: \langle 1, 4, 4 \rangle$	$\langle 3, 4, 4 \rangle, 2$	$\langle *, *, * \rangle, *$
$V_3: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$
$V_4: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$

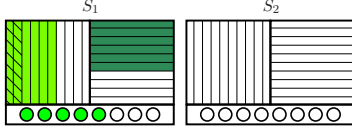


Figure 2: Next-Fit-Sharing: Allocating V_2

V_j	S_1	S_2
$\langle q_j^u, q_j^m, q_j^s \rangle$	$\langle C_1^u, C_1^m, C_1^s \rangle, s_j^1$	$\langle C_2^u, C_2^m, C_2^s \rangle, s_j^2$
$V_1: \langle 4, 4, 2 \rangle$	$\langle 4, 4, 8 \rangle, 0$	$\langle *, *, * \rangle, *$
$V_2: \langle 1, 4, 4 \rangle$	$\langle 3, 4, 4 \rangle, 2$	$\langle *, *, * \rangle, *$
$V_3: \langle 2, 4, 1 \rangle$	$\langle 1, 2, 3 \rangle, 3$	$\langle *, *, * \rangle, *$
$V_4: \langle *, *, * \rangle$	$\langle *, *, * \rangle, *$	$\langle *, *, * \rangle, *$

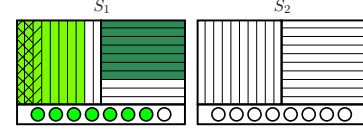


Figure 3: Next-Fit-Sharing: Allocating V_3

V_j	S_1	S_2
$\langle q_j^u, q_j^m, q_j^s \rangle$	$\langle C_1^u, C_1^m, C_1^s \rangle, s_j^1$	$\langle C_2^u, C_2^m, C_2^s \rangle, s_j^2$
$V_1: \langle 4, 4, 2 \rangle$	$\langle 4, 4, 2 \rangle, 0$	$\langle *, *, * \rangle, *$
$V_2: \langle 1, 4, 4 \rangle$	$\langle 4, 4, 2 \rangle, 2$	$\langle *, *, * \rangle, *$
$V_3: \langle 2, 4, 1 \rangle$	$\langle 1, 2, 3 \rangle, 3$	$\langle *, *, * \rangle, *$
$V_4: \langle 4, 4, 4 \rangle$	$\langle *, *, * \rangle, *$	$\langle 4, 6, 6 \rangle, 0$

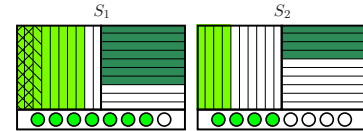


Figure 4: Next-Fit-Sharing: Allocating V_4

VM request V_j , the cloud provider determines if V_j can be packed onto the active server $S_i \in \mathcal{S} \setminus \bar{\mathcal{S}}$.

- If active server S_i has enough available capacity for every resource type to instantiate VM V_j while considering memory sharing in the amount of s_j^i , then V_j is packed onto server S_i (line 10 of Algorithm 1).
- Else, server S_i is closed, inactive server S_{i+1} is activated, and V_j is packed onto S_{i+1} (lines 4 to 8 of Algorithm 1).

The difference between Next-Fit-Sharing and the standard sharing-oblivious Next-Fit algorithm is that Next-Fit-Sharing accounts for page sharing within each server when allocating the incoming VMs; thereby reducing the memory utilization and implicitly the number of servers required to satisfy the user requests.

As an example, in Figure 1 through Figure 4, we show the execution of Next-Fit-Sharing for a given VM sequence. We consider servers that have the following resource capacities: 8 CPUs, 10 GB of RAM, and 100 GB of storage. We represent a server as a box with three rectangular sections that correspond to: (i) CPUs, represented by circles in the bottom section; (ii) memory resource, as the top left section, where each vertical stripe corresponds to 1 GB of RAM; and (iii) storage resource, as the top right section, where each horizontal stripe corresponds to 10 GB of storage. When a component within one of the server sections is colored, that component resource is interpreted as provisioned to a requesting VM.

According to Next-Fit-Sharing, only one server is active at a time until a VM cannot be instantiated on the server; at which point that server is closed and the next server is

activated. In Figure 1, a single VM request, V_1 , arrives and it is allocated to the first server, S_1 . After V_1 is allocated, the remaining S_1 resource capacities are 4 CPUs, 6 GB of RAM, and 80 GB of storage. In Figure 2, V_2 arrives and is allocated to the first server S_1 . As mentioned before, prior to the allocation, a staging process occurs whereby V_2 is fingerprinted for memory and then the sharing among V_2 and all VMs instantiated on S_1 is determined. In this example, the amount of sharing is 2 GB of memory pages, i.e., $s_2^1 = 2$. The process of sharing, in this case, is illustrated through the forward diagonal lines in the memory section for each server (Figure 2). Out of the 4 GB of RAM requested by V_2 , 2 GB are shared with V_1 and the other 2 GB are allocated separately from the rest of the VMs. Following V_2 being allocated, the remaining resource capacities of S_1 are 3 CPUs, 4 GB of RAM, and 40 GB of storage. Next, V_3 arrives and it is also allocated to S_1 (Figure 3). From the staging process, V_3 is found to share pages in the amount of 3 GB with all VMs instantiated on S_1 . The process of sharing, in this case, is illustrated through the backward diagonal lines in the memory section of each server, that is, 2 GB shared with V_1 and 1 GB shared with V_2 ; leaving 1 GB of RAM in isolation from the rest of the VMs. Following V_3 being allocated, the remaining resource capacities of S_1 are 1 CPU, 2 GB of RAM, and 10 GB of storage. Lastly, V_4 arrives (Figure 4), and due to S_1 not having enough available capacity to host V_4 , S_1 is closed, and S_2 is now activated and waiting for VMs to host. Since V_4 is the only VM on S_2 , there are no pages to share, thus the remaining capacities of S_2 are 4 CPUs, 6 GB of RAM, and 60 GB of storage.

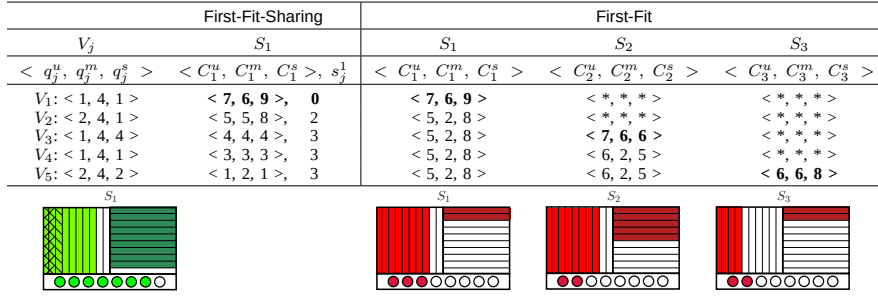


Figure 5: First-Fit-Sharing vs. First-Fit: VM Sequence Allocation.

B. First-Fit-Sharing Algorithm

The First-Fit-Sharing algorithm is given in Algorithm 2 and works as follows. Upon VM V_j request arrival, a search ensues to determine the first active server S_k from the set of active servers $\mathcal{S} \setminus \bar{\mathcal{S}}$ on which V_j can be packed.

- V_j is packed onto the first active server $S_k \in \mathcal{S} \setminus \bar{\mathcal{S}}$ that has enough available capacity for every resource type to instantiate VM V_j while considering memory sharing in the amount of s_j^k , (lines 5 to 8 of Algorithm 2).
- If V_j cannot be packed onto any of the active servers, then server S_{i+1} is activated and V_j is packed onto it (lines 9 to 13 of Algorithm 2).

Algorithm 2 First-Fit-Sharing

```

1: Input: VM instance arrival ( $V_j$ )
2:  $flag \leftarrow 1$ 
3: for all  $k : S_k \in \mathcal{S} \setminus \bar{\mathcal{S}}$  do
4:   if ( $flag$ ) then
5:     if  $(C_k^m - q_j^m + s_j^k \geq 0)$  and  $(C_k^u - q_j^u \geq 0)$ 
       and  $(C_k^s - q_j^s \geq 0)$  then
6:        $S_k \leftarrow S_k \cup \{V_j\}$ 
7:        $i \leftarrow k$ 
8:        $flag \leftarrow 0$ 
9: if ( $flag$ ) then
10:   $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \{S_{i+1}\}$ 
11:  Activate server  $S_{i+1}$ 
12:   $S_{i+1} \leftarrow S_{i+1} \cup \{V_j\}$ 
13:   $i \leftarrow i + 1$ 
14:  $[C_i^m, C_i^u, C_i^s] \leftarrow [C_i^m, C_i^u, C_i^s] - [q_j^m - s_j^i, q_j^u, q_j^s]$ 

```

The difference between First-Fit-Sharing and the standard sharing-oblivious First-Fit algorithm, is that First-Fit-Sharing accounts for page sharing within each server when allocating the incoming VMs; thus reducing the total memory utilization and implicitly the number of active servers required to satisfy the user requests. In Figure 5, we provide an example in which we compare the First-Fit-Sharing and the classical sharing-oblivious First-Fit algorithm on a sequence of VMs requesting instantiation. This example shows an instance where the First-Fit algorithm uses two servers more than the First-Fit-Sharing algorithm, which is a big savings of resources for this small scale example. We expect significant resource and server savings for larger scale systems as we show in Section IV.

C. Best-Fit-Sharing Algorithm

In order to design the Best-Fit-Sharing, we introduce the *server resource scarcity* metric which characterizes the scarcity of aggregate resources at a given server relative to the requested resources by a VM. The classical version of the Best-Fit packing algorithm places a new item into the bin with the least remaining current capacity according to one dimension, i.e., the size of the item in one dimension. Since the VM packing problem considers multiple resource requirements, we have to consider all required resources and available capacities when determining the appropriate server for allocating the requesting VM. We define the *server resource scarcity* metric as follows:

$$E_j^k = \sum_{r \in R} \frac{q_j^r}{C_k^r} + \frac{q_j^m - s_j^k}{C_k^m} \quad (3)$$

The metric characterizes the scarcity of the aggregate resources of the k^{th} server relative to the resource requirements of V_j .

Algorithm 3 Best-Fit-Sharing

```

1: Input: VM instance arrival ( $V_j$ )
2:  $flag \leftarrow 0$ 
3:  $\bar{E} \leftarrow 0$ 
4:  $idx \leftarrow 0$ 
5: for all  $k : S_k \in \mathcal{S} \setminus \bar{\mathcal{S}}$  do
6:   if  $(C_k^m - q_j^m + s_j^k \geq 0)$  and  $(C_k^u - q_j^u \geq 0)$  and  $(C_k^s - q_j^s \geq 0)$ 
     then
7:     for all  $r \in R$  do
8:        $E_j^k \leftarrow E_j^k + \frac{q_j^r}{C_k^r}$ 
9:      $E_j^k \leftarrow E_j^k + \frac{q_j^m - s_j^k}{C_k^m}$ 
10:    if  $\bar{E} < E_j^k$  then
11:       $idx \leftarrow k$ 
12:       $\bar{E} \leftarrow E_j^k$ 
13:       $flag \leftarrow 1$ 
14: if ( $flag$ ) then
15:    $i \leftarrow idx$ 
16:    $S_i \leftarrow S_i \cup \{V_j\}$ 
17: else
18:    $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \{S_{i+1}\}$ 
19:   Activate server  $S_{i+1}$ 
20:    $S_{i+1} \leftarrow S_{i+1} \cup \{V_j\}$ 
21:    $i \leftarrow i + 1$ 
22:  $[C_i^m, C_i^u, C_i^s] \leftarrow [C_i^m, C_i^u, C_i^s] - [q_j^m - s_j^i, q_j^u, q_j^s]$ 

```

We use the metric in the design of the Best-Fit-Sharing algorithm, given in Algorithm 3.

The Best-Fit-Sharing algorithm works as follows. Upon the arrival of VM request V_j , a search ensues to determine the active server S_k in which V_j can be packed. The search is performed on the set of all active servers i.e., $\mathcal{S} \setminus \bar{\mathcal{S}}$.

- V_j is packed onto the first active server $S_k \in \mathcal{S} \setminus \bar{\mathcal{S}}$ that has enough available capacity for every resource type to instantiate VM V_j while considering memory sharing in the amount of s_j^k , and has the largest value for the server resource scarcity metric E_j^k . (lines 6 to 13 of Algorithm 3).
- If V_j cannot be packed onto any of the active servers, then server S_{i+1} is activated and V_j is packed onto it (lines 17 to 21 of Algorithm 3).

D. Worst-Fit-Sharing Algorithm

Since Worst-Fit-Sharing can be viewed as the dual of Best-Fit-Sharing and thus, it has a structure that is very similar to that of Best-Fit-Sharing, we will not provide a formal algorithmic description of it. The only difference between the two algorithms is that Worst-Fit-Sharing allocates the new VM request to the server that has the minimum server resource scarcity value, that is, the comparison $\tilde{E} < E_j^k$ in Best-Fit-Sharing (Line 10 of Algorithm 3) is changed to $\tilde{E} > E_j^k$ for the Worst-Fit-Sharing algorithm.

The Worst-Fit-Sharing algorithm works as follows. Upon the arrival of VM request V_j , a search ensues to determine the active server S_k in which V_j can be packed. The search is performed on the set of all active servers i.e., $\mathcal{S} \setminus \bar{\mathcal{S}}$.

- V_j is packed onto the first active server $S_k \in \mathcal{S} \setminus \bar{\mathcal{S}}$ that has enough available capacity for every resource type to instantiate VM V_j while considering memory sharing in the amount of s_j^k , and has the smallest value for the server resource scarcity metric, E_j^k .
- If V_j cannot be packed onto any of the active servers, then server S_{i+1} is activated and V_j is packed onto it.

E. Run-time Complexity

We analyze the runtime complexity of each of our proposed sharing-aware online algorithms. In order to have a more accurate run-time analysis of the algorithms we have to include in the analysis, the running-time of computing the amount of memory sharing, s_j^k , among the requested V_j and the VMs hosted at server S_k ; calculated based on the V_j memory fingerprint determined on the staging server, as explained in Section II.

In the case of Next-Fit-Sharing, allocating a VM V_j requires searching for page sharing opportunities relative to V_j and only one active server S_k . The search for shared pages consists of comparing each page required by V_j against the set of pages instantiated on server S_k . If we denote by n the maximum between the number of unique pages on server S_k and the number of pages required by V_j , then determining the number of shared pages requires $O(n^2)$ time. The run

time complexity of Next-Fit-Sharing without including the determination of the amount of memory sharing is $O(1)$. Therefore, the total run time of Next-Fit-Sharing is $O(n^2)$.

First-Fit-Sharing, Best-Fit-Sharing, and Worst-Fit-Sharing require searching for page sharing relative to the current request, V_j , and the entire set of active servers, $\mathcal{S} \setminus \bar{\mathcal{S}}$. If we denote the number of active servers prior to hosting V_j by m , then the run time for all three algorithms is $O(mn^2)$.

IV. EXPERIMENTAL RESULTS

We perform extensive experiments investigating the performance of our proposed sharing-aware online algorithms for solving SA-OVMP by metrics such as the average number of active servers, resource utilization, *Aggregate-Min* ratio and *Max-Min* ratio. In the following, we describe the experimental setup and analyze the results.

A. Experimental Setup

We compare our proposed online sharing-aware algorithms against each other and against their classical, sharing-oblivious counterparts. All these algorithms are online algorithms, that is no information about incoming VM requests is known beforehand. We perform the simulations on a 2.4 GHz Intel Core[®] i7-3630 QM CPU 64-bit system. All simulations are implemented in C++ and are compiled with GCC v4.9.2.

In addition to the classical sharing-oblivious online algorithms First-Fit and Next-Fit we also consider a sharing-oblivious version of the Random-Fit algorithm. Random-Fit is implemented similarly to First-Fit with the exception that once a server has been discovered that it meets the VM resource requirements, the VM is allocated and the ordering of the servers is changed by randomly permuting them. When the next request arrives, the servers will have a different ordering.

In our experiments, we consider a cloud provider that has 500 identical servers available for hosting VMs. Each of these servers has the following resource capacities: (i) 8 CPUs; (ii) 10 GB of memory; and (iii) 100 GB of disk space. We execute 1,000 simulations which consist of allocating 500 VMs per simulation. We simulate 80 unique types of VM instances with resource requests in the following ranges: CPUs (in units of 1, 2 or 4 CPUs); memory (from 1 GB to 4 GB RAM in increments of 500 MB), and storage (in units of 1, 2 or 4 GB of disk space). These ranges were chosen to reflect the current VM instance specifications used by cloud service providers such as Amazon EC2. The 500 VMs for each simulation are selected according to a random uniform distribution from the set of 80 unique types of VMs.

The sharing of memory pages for our experiment is induced by the parameters of our 80 unique VM types. Each VM V_j in our experiment is characterized in addition to the CPU, memory, and storage requirements, by the set of applications that need to be executed on it, the operating system, and a memory sharing parameter, s_j^k . We consider three types of operating systems (A, B, and C) and two types of application bundles for generating the VM requests. The memory sharing parameter is assigned in amounts ranging from 500 MB and

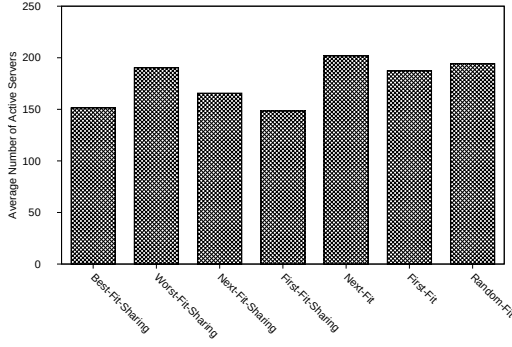


Figure 6: Average Number of Active Servers.

up to q_j^m GB in increments of 500 MB, where each 500 MB corresponds to a given number of memory pages. To show how the memory sharing parameter is determined, we consider an example VM of a type selected from our set of 80 unique VM types, requiring 1 CPU, 2 GB of memory, 2 GB of storage, one type of bundle of applications that it needs to execute, and a given type of operating system. For this set of requests we generate four values for the sharing parameter 0.5 GB, 1 GB, 1.5 GB, and 2 GB (i.e., up to the amount of memory request q_j^m), thus leading to four types of VM instances with four different sharing potentials. If for example, we consider a VM request that has the sharing parameter 0.5 GB, then if another VM is hosted on the same server utilizing the same applications and/or operating system, then 0.5 GB would be shared between the VMs on the server and the requested VM.

B. Analysis of Results

We now compare our proposed online sharing-aware algorithms for SA-OVMP against each other and against other online algorithms, as previously described.

In Figure 6, we plot the average number of active servers used by all online algorithms in our experiment. Provided our simulation parameters, First-Fit-Sharing activates 149 servers in average, representing the minimum average number of servers relative to the other algorithms. Best-Fit-Sharing performs slightly worse than First-Fit-Sharing activating 152 servers. First-Fit performs the best among the sharing-oblivious algorithms, activating 188 servers. On average, the proposed online sharing-aware algorithms outperform their sharing-oblivious counterparts and Random-Fit. These results show that by employing our proposed sharing-aware algorithms a cloud provider can accommodate the requests using a significantly smaller number of servers than when employing the sharing-oblivious algorithms, and thus, significantly reducing the energy costs necessary for serving the VM requests.

In Figure 7, we plot the average percent of unused resources over all active servers throughout our simulations. For an online algorithm to be effective in allocating VMs with multiple resource requirements, it should use as much of the available server capacities as possible, for every type of resource. If a cloud service provider instantiates multiple in-demand VMs

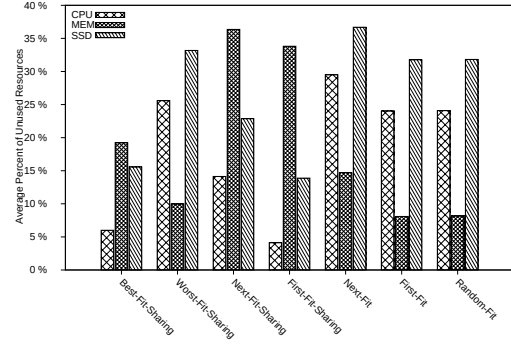


Figure 7: Average Percent of Unused Resources.

that absorb a significant amount of CPU resource, for instance, then once that resource has been exhausted, the utility of the other resources is zero since VMs have requirements with respect to at least three resources, therefore making the server unavailable for new instantiations for a given period of time. Provided our simulation parameters, Best-Fit-Sharing consumes a considerable amount of all three types of resources when compared to the other algorithms. In particular, Best-Fit-Sharing exhibits an average of approximately 6% unused CPU resource, approximately 19% unused memory resource, and approximately 15% unused disk space. The next effective online algorithm is First-Fit-Sharing, which exhibits an average of approximately 4% of unused CPU resource, approximately 33% unused memory resource, and approximately 14% unused disk space. Best-Fit-Sharing performs better than First-Fit-Sharing, since Best-Fit-Sharing searches over all active servers and finds the best server to allocate to, based on the server resource scarcity metric. First-Fit-Sharing may miss a good allocation due to its allocation policy; thus, leading to more variations in unused resources than in the case of Best-Fit-Sharing. The analysis of Figure 7 is important for cloud service providers as they gain insight into which online algorithm may over- or under-utilize server resources.

In Figure 8, we plot the *Aggregate-Min* ratios for all online algorithms used in our experiment. The *Aggregate-Min* ratio is the total number of servers used throughout the experiment for a specific online algorithm over the minimum total number of servers determined by the best performing algorithm. We consider the *Aggregate-Min* ratio as an experimental metric which reflects how far the performance of an online algorithm is to the best performing online algorithm in the experiments, where performances of both algorithms are obtained on the same VM sequences throughout the experiment. Provided our simulation parameters, First-Fit-Sharing uses the minimum total number of servers. In this case, the *Aggregate-Min* ratio for First-Fit-Sharing is 1.00 and the next best performing algorithm is Best-Fit-Sharing with an *Aggregate-Min* ratio of 1.0201. Next-Fit-Sharing is the next best performing online algorithm with an *Aggregate-Min* ratio of 1.1167.

In Figure 9, we plot the *Max-Min* ratios for all online algorithms in our experiment. The *Max-Min* ratio is a metric that

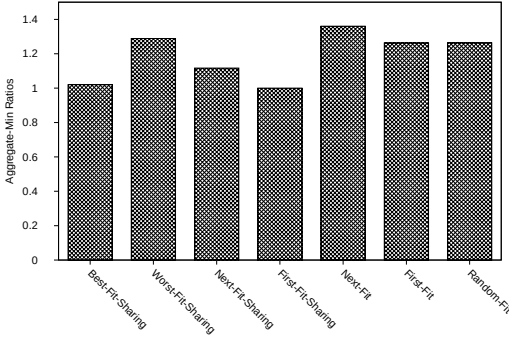


Figure 8: Aggregate-Min Ratios.

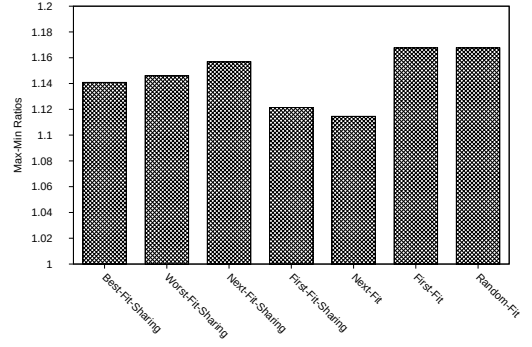


Figure 9: Max-Min Ratios.

characterizes the performance of each individual online algorithm. The ratio provides insight into an algorithm's robustness and is defined as the ratio of the number of servers used by the algorithm when provided the worst sequence of VM requests considered in our simulations, over the number of servers used when provided with the best sequence of VM requests considered in our simulations (i.e., the sequence for which the algorithm uses the smallest number of servers). Provided our simulation parameters, Next-Fit has the lowest *Max-Min* ratio of 1.1145833, which implies that the performance of Next-Fit is the most robust against VM sequence variations among all the considered online algorithms. The average server count for Next-Fit under worst and best VM sequences tend to both be greater than other algorithm's server count under worst and best VM sequences. Therefore, the variance between the server count for Next-Fit under a worst VM sequence allocation and under a best VM sequence allocation will be the least on average when expressed as a ratio. The next lowest *Max-Min* ratio of 1.1214286 is obtained by First-Fit-Sharing, followed by Best-Fit-Sharing with a *Max-Min* ratio of 1.1408451.

V. CONCLUSION

We designed a family of online sharing-aware algorithms for solving the sharing-aware online VM packing (SA-OVMP) problem. We analyzed the runtime of each proposed sharing-aware online algorithm and investigate their performance by performing an extensive set of experiments. The experimental results showed that our proposed sharing-aware online algorithms activated a smaller average number of servers relative to their sharing-oblivious counterparts, reduced server utilization of CPUs, memory and storage, and thus, allowing for VMs to be packed using fewer servers. In future work, we plan on investigating the theoretical bounds for our sharing-aware online algorithms and quantifying their behavior through relative worst order ratios. We also plan to deploy the proposed algorithms on an experimental cloud testbed and analyze their performance.

ACKNOWLEDGMENT

This research was supported in part by NSF grants DGE-0654014 and CNS-1116787.

REFERENCES

- [1] CISCO, Cisco Global Cloud Index: Forecast and Methodology. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html
- [2] G. Mišós, D. G. Murray, S. Hand, and M. A. Fetterman, "Satori: Enlightened page sharing," in *Proc. 2009 USENIX Annual Technical Conf.*, 2009.
- [3] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. D. Corner, "Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers," in *Proc. ACM Intl. Conf. on Virtual Execution Environments*, 2009, pp. 31–40.
- [4] D. Gupta, S. Lee, M. Vrabie, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, "Difference engine: Harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, Oct. 2010.
- [5] M. Sindelar, R. Sitaraman, and P. Shenoy, "Sharing-aware algorithms for virtual machine colocation," in *Proc. 23rd ACM Symp. on Parallelism in Algorithms and Architectures*, 2011, pp. 367–378.
- [6] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *Microsoft Research - Technical Report*, 2011.
- [7] W. Song, Z. Xiao, Q. Chen, and H. Luo, "Adaptive resource provisioning for the cloud using online bin packing," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2647–2660, Nov 2014.
- [8] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proc. 26th ACM Symp. on Parallelism in Algorithms and Architectures*, 2014, pp. 2–11.
- [9] S. Kamali and A. López-Ortiz, "Efficient online strategies for renting servers in the cloud," in *SOFSEM 2015: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, vol. 8939, pp. 277–288.
- [10] T. Carli, S. Henriot, J. Cohen, and J. Tomasik, "A packing problem approach to energy-aware load distribution in clouds," *CoRR*, vol. abs/1403.0493, 2014. [Online]. Available: <http://arxiv.org/abs/1403.0493>
- [11] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds," in *Proc. IEEE INFOCOM*, 2012, pp. 2861–2865.
- [12] C. Kleinschweber, A. Reinefeld, and T. Schütt, "Qos-aware storage virtualization for cloud file systems," in *Proc. 1st ACM Intl. Workshop on Programmable File Systems*, 2014, pp. 19–26.
- [13] S. Rampersaud and D. Grosu, "A sharing-aware greedy algorithm for virtual machine maximization," in *Proc. 13th IEEE Intl. Symp. on Network Computing and Applications*, Aug 2014.
- [14] —, "A multi-resource sharing-aware approximation algorithm for virtual machine maximization," in *Proc. 3rd IEEE Intl. Conf. on Cloud Engineering*, March 2015.
- [15] Y. Azar, I. R. Cohen, S. Kamara, and B. Shepherd, "Tight bounds for online vector bin packing," in *Proc. 45th Annual ACM Symp. on Theory of Computing*, 2013, pp. 961–970.
- [16] J. Boyar and L. M. Favrholdt, "The relative worst order ratio for online algorithms," *ACM Trans. Algorithms*, vol. 3, no. 2, May 2007.