2014 IEEE 13th International Symposium on Network Computing and Applications

A Sharing-Aware Greedy Algorithm for Virtual Machine Maximization

Safraz Rampersaud
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: safraz@wayne.edu

Daniel Grosu
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: dgrosu@wayne.edu

***Abstract*—Service providers face multiple challenges in hosting an increasing number of virtual machine (VM) instances. Minimizing the utilization of system resources while maximizing the potential for profit are among the most common challenges. Recent studies have investigated memory reclamation techniques focused on virtual technologies, specifically page sharing, for minimizing the utilization of system resources. In this paper, we address the problem of sharing-aware VM maximization in a general sharing model which has as objective finding a subset of VMs that can be hosted by a server with a given memory capacity such that the total profit derived from hosting the subset of VMs is maximized. The sharing-aware VM maximization allocation problem has been shown to be NP-hard. Therefore, we design a greedy approximation algorithm for solving it. We determine the approximation ratio of our greedy algorithm and perform extensive experiments to investigate its performance against other VM allocation algorithms.**

***Index Terms*—virtual machine, resource allocation, greedy algorithms.**

I. Introduction

Virtualization, the process of abstracting a software layer which decouples the physical hardware from the operating system to deliver greater resource utilization and flexibility [1], serves as a means to increase productivity, lower power consumption, reduce hardware installation, and overall, minimize the need for increasing the resource capacity to meet the demand [2]. The application of virtualization technologies is ubiquitous in data centers around the world which must consider operational costs and guarantee fast delivery of a variety of profitable services. Specifically, the service providers must ensure the efficiency of their virtualized services in a competitive environment where fast entry to market, technology advancement, and service pricing differentials can separate sustaining providers from antiquated ones. Proprietary virtualization platforms, such as VMWare's ESX Suite, Microsoft's Hyper-V, and IBM's PowerVM, differ in their methods of operations (e.g., full-, para- and hardware assisted-virtualization), overhead, and available number of guest OS hosting capacities, among other features. Open-source alternatives, such as Xen, KVM, and Linux-VServer, offer comparable features and operations to the proprietary platforms, while being supported by a large online community. Moreover, open-source virtualization systems such as Xen [3] have improved the user experience by implementing safe resource management strategies without losing performance and/or functionality.

Virtualization has undergone a significant evolution spanning approximately half a century. Innovations within virtualization technologies were initially focused on overcoming the limitations of third-generation computing architectures [4]. Within this context, virtualization solved the problem of protecting non-privileged references to end users when multiple end users attempted to access non-privileged instructions through a privileged mode on the base machine [4]. Invocation of a software layer to access the non-privileged instructions, known at the time as the *privileged software nucleus*, suffered from single access to the non-privileged references limiting the potential for multiple users. Hence, virtualization was born out of these limitations and fulfilled the opportunity to replicate the privileged and non-privileged instruction sets from the base machine, known as the *host*, for multiple end users through a transformed software layer referred to as a *hypervisor*.

Minimizing resource consumption has been a key driver in the overall advancement of virtualization technologies. Memory reclamation techniques such as ballooning, hypervisor swapping, memory compression, and page sharing all attempt to efficiently utilize virtual machine (VM) memory [5]. Page sharing creates new challenges in the development of algorithms which allocate VMs onto server resources. The problem of allocating VMs onto a single server to maximize the profit, where the profit is the sum of the profits derived from hosting each individual VM, is equivalent to the knapsack problem in which each VM is an object and the number of memory pages required to host each VM is the object's weight. Therefore, each VM can be treated as a distinct object having a weight and a profit given by the profit derived from hosting it. As a result of this equivalence, knapsack heuristic algorithms can be applied to solve the above VM allocation problem when page sharing is not considered. When the sharing of pages among the VMs is considered, the problem of VM profit maximization is no longer equivalent to the knapsack problem. Existing knapsack algorithms will produce less than the maximum profit due to not allocating additional VMs on the extraneous server resources which become available when VM pages are shared; resulting in loss of profit. Therefore, new algorithms for VM maximization that take into account

978-1-4799-5393-6/14 $31.00 © 2014 IEEE
DOI 10.1109/NCA.2014.24

the sharing of pages among VMs must be developed.

A. Our Contribution

We address the problem of sharing-aware VM maximization in a general sharing model which has as objective finding a subset of VMs that can be hosted by a server with a given memory capacity such that the total profit derived from hosting the subset of VMs is maximized. This problem has been shown to be NP-hard [6]. Therefore, we design a greedy approximation algorithm based on a new efficiency metric which considers both profit-seeking and page sharing opportunities in the VM allocation process. We determine the approximation ratio of our greedy algorithm that solves the sharing-aware VM maximization problem in the general sharing model, a model that does not assume any hierarchical or other structured form of sharing. We perform extensive experiments to evaluate the performance of our greedy algorithm against other VM allocation algorithms.

B. Related Work

The sharing-aware VM maximization problem has been introduced by Sindelar *et al.* [6]. Their main contributions lie in the development of hierarchical sharing models for VM colocation for both the VM maximization and packing problems. They were the first to propose and investigate algorithms for solving the sharing-aware VM maximization problem. Their research is the closest to our research. Our research on the sharing-aware VM maximization problem focuses on the general sharing model which differs from the shared hierarchical models investigated by Sindelar *et al.* [6].

The sharing-aware VM maximization problem has been shown to be NP-hard [6]. Thus, solving it optimally is not feasible and we have to resort to approximation algorithms, more specifically greedy algorithms. Greedy algorithms have been extensively investigated for different classical problems such as the knapsack [7], subset-sum, partition [8], as well as, facility location [9]. Greedy algorithms for VM provisioning and allocation in clouds have been investigated by Zaman and Grosu [10], who designed combinatorial auction-based mechanisms. Nejad *et al.* [11] designed a family of truthful greedy heuristic mechanisms for dynamic VM provisioning. Other research on greedy heuristics for VM provisioning focused on minimizing bandwidth-constraint VM placement in data centers [12] and on minimizing power consumption [13]. All these works focused on designing algorithms for provisioning VMs on multiple physical machines within a cloud computing system, and for allocation of VMs to users. Our work focuses on developing algorithms that maximize the profit derived from hosting VMs on a single physical machine that can be employed in making decisions at the physical machine level and work in conjunction with higher level resource management algorithms such as the ones discussed above.

Much of the work on page sharing focused on developing page sharing systems. Bugnion *et al.* [14] proposed the transparent page sharing technique for minimizing redundancy and memory overhead. Commercial systems such as VMWare's ESX Server [15] enable transparent page sharing in addition to other memory reclamation techniques [5]. Wood *et al.* [16] proposed *Memory Buddies*, a sharing-aware VM memory allocation system which uses the VMWare ESX Server to identify page sharing opportunities. This is achieved by employing hashing algorithms that capture the potential for sharing between multiple VMs. The open source Xen hypervisor [3], has incorporated page sharing in Versions 4.0 and above for Hardware Virtual Machines (HVM) [17]. Gupta *et al.* [18] developed the *Difference Engine* system which incorporates sub-page sharing, i.e., sharing pages that are nearly identical, and uses compression techniques for pages that are not similar, thereby further reducing the overall memory footprint. Our work focuses on developing sharing-aware VM allocation algorithms that maximize the profit obtained from hosting the VMs and take into account page sharing.

C. Organization

The rest of the paper is organized as follows. In Section II, we describe the sharing-aware VM maximization problem. In Section III, we present the design of our proposed efficiency metric and our greedy algorithm for the sharing-aware VM maximization problem. In Section IV, we characterize the properties of the proposed greedy algorithm. In Section V, we evaluate our greedy algorithm against other VM allocation algorithms by extensive experiments. In Section VI, we summarize our results and present possible directions for future research.

II. Sharing-Aware VM Maximization

In this section, we introduce the SAVMM (Sharing-Aware Virtual Machine Maximization) problem as it applies to a service provider resource environment.

We assume that a service provider maintains a server Ω, and a library Π of all memory pages required for each service it offers. Thus, the provider can identify and manage all memory pages required by a VM. We denote by π_i, the i-th memory page under the provider's management. Library Π is comprised of N distinct pages, i.e., $\Pi = \bigcup_{i=1}^{N} \{\pi_i\}$.

Each VM instance requires a set of memory pages which virtualizes a service offered by the provider. We denote by V_j, the VM instance j, by Λ_j, the set of indices of pages required by V_j, and by π_i^j, the i-th memory page required by VM V_j. We denote by $\mathcal{V}$, the set of "offline" VM instances that are possible candidates for allocation and hosting on server Ω. Given this setup, we define the SAVMM problem as follows:

> SAVMM problem: Given a set of M "offline" VMs $\mathcal{V}$ with each VM V_j yielding a profit of p_j, determine a subset $\mathcal{V}^H \subset \mathcal{V}$ of VMs that can be allocated on the server, considering the memory capacity C of the server and the sharing of pages within library Π, such that the total profit, $P = \sum_{j:V_j \in \mathcal{V}^H} p_j$, obtained by the provider is maximized.

The SAVMM problem may appear similar to the standard knapsack problem [7], but it is not the same, because the items (VMs) in the SAVMM problem are shared, while the items in

Table I: Notation.

Expression	Description
Π	Set of pages under provider's management.
N	Number of memory pages under provider's management.
$\mathcal{V}$	Set of offline VMs.
M	Number of offline VMs.
$\mathcal{V}^H$	Subset of VMs maximizing provider's profit, $\mathcal{V}^H \subset \mathcal{V}$.
V_j	Virtual machine j
π_i	The i-th memory page under provider's management.
π_i^j	The i-th memory page requested by VM V_j.
p_j	Profit generated from allocating VM V_j.
Λ_j	Set of indices of pages requested by VM V_j.
Ω	Provider's server resource.
C	Memory capacity of server resource Ω.
k	Iteration number.
$\mathcal{E}_j^k$	Efficiency metric of VM V_j at iteration k.
$\mathcal{S}_j^k$	Number of pages VM V_j shares with Ω at iteration k.

the standard knapsack problem are not. Server Ω can host all the VMs in $\mathcal{V}$, if all the VMs in the set share the same pages and the total number of allocated pages does not exceed the capacity C of the server. The notation we use throughout the paper is summarized in Table I.

III. Greedy Approximation Algorithm (G-SAVMM)

In this section, we present the design of our greedy algorithm for solving the SAVMM problem. The main idea used in the design of our greedy algorithm is to order the candidate VMs according to a metric which characterizes their potential for profit and page-sharing and then allocate them one by one according to the greedy order. The greedy metric and the greedy order is updated after allocating each VM. This represents an iteration in the greedy allocation process and will be denoted by k.

We first introduce the proposed metric we use in our greedy algorithm to establish the greedy order among the candidate VMs. At every iteration k, we order the candidate VMs, $V_j \in \mathcal{V}$, according to an efficiency metric, $\mathcal{E}_j^k$, defined as follows:

$$\mathcal{E}_j^k = \frac{p_j}{\sqrt{K_j - S_j^k + 1}}. \quad (1)$$

where j is the index corresponding to VM V_j, K_j is the number of pages required by VM V_j (i.e., $K_j = |\Lambda_j|$), and S_j^k is the number of shared pages between VM V_j and the VMs that are already allocated to the server.

The efficiency metric $\mathcal{E}_j^k$ represents the relative value of allocating VM V_j onto Ω by considering the profit p_j and the potential for sharing pages characterized by S_j^k, where k corresponds to the current greedy iteration. Prior to allocating the first VM onto Ω (i.e., at iteration $k = 0$), the efficiency metric for the "offline" set $\mathcal{V}$ of VMs is calculated using S_j^0 determined relative to the number of shared pages within all the VMs in $\mathcal{V}$ and not relative to the VMs that are allocated on the server. Once a VM has been selected and allocated (i.e., for all iterations $k > 0$) then $\mathcal{E}_j^k$ is calculated using $\mathcal{S}_j^k$, the number of shared pages between VM V_j and the VMs that are already allocated onto the server. As k increases and VMs are allocated onto Ω, we have $\mathcal{S}_j^k \leq \mathcal{S}_j^{k+1}$, that is $\mathcal{S}_j^k$ monotonically increases with k, for $k > 0$.

Since $\mathcal{E}_j^k$ needed to be well defined for all possible cases, we add 1 to the denominator. The reason for this is that, if VM V_j shares all its pages with another VM already allocated onto Ω, (i.e., $K_j = S_j^k, \forall k$), and if we do not consider adding 1 to the denominator of $\mathcal{E}_j^k = \frac{p_j}{\sqrt{K_j - S_j^k}}$, then the efficiency metric would produce an indeterminate value. We also reduce the magnitude of the sharing potential in the efficiency metric against the profit by applying a square root to the denominator. Profit has the largest effect when calculating the efficiency metric and therefore we want to capture as much effect as possible, while still allowing for the influence of page sharing. Similar metrics to our efficiency metric have been experimented with in studies focusing on the knapsack problem [7] and have led to good approximation ratios.

The G-SAVMM algorithm for solving the SAVMM problem is presented in Algorithm 1. G-SAVMM consists of two phases: (i) a pre-processing phase, for $k = 0$ (Lines 2 through 28); and, (ii) a greedy allocation phase, for $k > 0$ (Lines 29 through 49). The input of G-SAVMM is an "offline" set of VMs $\mathcal{V}$. G-SAVMM determines the set $\mathcal{V}^H$ of VMs allocated onto the server, which is an approximate solution to the SAVMM problem.

In the pre-processing phase, G-SAVMM scans every VM V_j to identify its required pages, denoted by π_i^j. activePage() (Line 8) is a function that returns 1, if page π_i^j is requested, and 0, otherwise. For every active page π_i^j the algorithm increments the variable K_j, the number of pages required by VM V_j, and also A_i, the number of page π_i occurrences among all VMs in $\mathcal{V}$ (Lines 6 through 10). After calculating A, it determines the page from $\mathcal{V}$ that has the maximum number of requests, identified by index $\tilde{i}$ (Line 11). If a VM requests page $\pi_{\tilde{i}}$, that VM will be placed in the subset $\mathcal{V}^H$ (Lines 12 through 14). Then the algorithm calculates S_j^0, the number of shared pages among the VMs in $\mathcal{V}^H$, by identifying the active pages where $A_i > 1$, implying more than one VM is requesting page i (Lines 15 through 18). The efficiency metric (Eq. 1) is then calculated for all VMs in subset $\mathcal{V}^H$ (Lines 19 and 20). Once the VM with the largest efficiency value, denoted by $V_{\tilde{j}}$, is identified (Line 21), the server capacity C is reduced by the number of pages $K_{\tilde{j}}$ in $V_{\tilde{j}}$ (Line 22). Following the server capacity reduction, the subset $\mathcal{V}^H$ is modified by eliminating all VMs with the exception of VM $V_{\tilde{j}}$ (Line 23) and then VM $V_{\tilde{j}}$ is removed from $\mathcal{V}$ (Line 24). Following the allocation of VM $V_{\tilde{j}}$, every requested page $\pi_i^{\tilde{j}}$ is identified, and π_i is activated on the server resource through a function called activate() (Lines 25 through 27). The activate() function implements the actions that need to be performed in order to make a page active on the server. The implementation of this function is platform specific and is out of the scope of this study. The pre-processing phase is completed with an update of the iteration number k to 1 (Line 28).

The greedy allocation phase of G-SAVMM, (i.e., iteration $k > 0$), is similar to the pre-processing phase ($k = 0$). At the

Algorithm 1 G-SAVMM: Greedy Algorithm for SAVMM

1: **Input:** Set of offline VM instances ($\mathcal{V}$)
2: {*Phase I: Pre-processing*}
3: $\mathcal{V}^H \leftarrow \emptyset$
4: $A \leftarrow \mathbf{0}$
5: $\tilde{i}, \tilde{j}, k \leftarrow 0$
6: **for** $i = 1, \ldots, N$ **do**
7: **for** $j = 1, \ldots, |\mathcal{V}|$ **do**
8: **if** (activePage(π_i^j)) **then**
9: $A_i = A_i + 1$
10: $K_j = K_j + 1$
11: $\tilde{i} = \text{argmax}_i\{A_i\}$
12: **for** $j = 1, \ldots, |\mathcal{V}|$ **do**
13: **if** (activePage($\pi_{\tilde{i}}^j$)) **then**
14: $\mathcal{V}^H = \mathcal{V}^H \cup \{V_j\}$
15: **for all** $j \in \mathcal{V}^H$ **do**
16: **for** $i = 1, \ldots, N$ **do**
17: **if** ($A_i > 1$) & (activePage(π_i^j)) **then**
18: $S_j^0 = S_j^0 + 1$
19: **for all** $j \in \mathcal{V}^H$ **do**
20: $\mathcal{E}_j^0 = \frac{p_j}{\sqrt{K_j - S_j^0 + 1}}$
21: $\tilde{j} = \text{argmax}_j\{\mathcal{E}_j^0\}$
22: $C = C - K_{\tilde{j}}$
23: $\mathcal{V}^H = \mathcal{V}^H \cap \{V_{\tilde{j}}\}$
24: $\mathcal{V} = \mathcal{V} \setminus \{V_{\tilde{j}}\}$
25: **for** $i = 1, \ldots, N$ **do**
26: **if** (activePage($\pi_i^{\tilde{j}}$)) **then**
27: activate(π_i)
28: $k \leftarrow 1$
29: {*Phase II: Greedy allocation*}
30: **while** ($C > 0$) & ($|\mathcal{V}| > 0$) **do**
31: $flag \leftarrow 1$
32: **for** $i = 1, \ldots, N$ **do**
33: **for** $j = 1, \ldots, |\mathcal{V}|$ **do**
34: **if** (activePage(π_i^j)) & (activePage(π_i)) **then**
35: $\mathcal{S}_i^k = \mathcal{S}_i^k + 1$
36: **for** $j = 1, \ldots, |\mathcal{V}|$ **do**
37: $\mathcal{E}_j^k = \frac{p_j}{\sqrt{K_j - S_j^k + 1}}$
38: $\tilde{j} = \text{argmax}_j\{\mathcal{E}_j^k\}$
39: **if** $C - (K_{\tilde{j}} - \mathcal{S}_{\tilde{j}}^k) < 0$ **then**
40: $flag \leftarrow 0$
41: $\mathcal{V} = \mathcal{V} \setminus \{V_{\tilde{j}}\}$
42: **if** ($flag$) **then**
43: $\mathcal{V}^H = \mathcal{V}^H \cup \{V_{\tilde{j}}\}$
44: $\mathcal{V} = \mathcal{V} \setminus \{V_{\tilde{j}}\}$
45: $C = C - (K_{\tilde{j}} - \mathcal{S}_{\tilde{j}}^k)$
46: **for** $i = 1, \ldots, N$ **do**
47: **if** (activePage($\pi_i^{\tilde{j}}$)) **then**
48: activate(π_i)
49: $k = k + 1$
50: $\Omega \leftarrow \mathcal{V}^H$
51: **exit**

beginning of the greedy phase, a test is performed to ensure that server capacity C is never exceeded and that there is at least one VM in $\mathcal{V}$ (Line 30). The differences between the two phases consists on how sharing is checked. In the first phase, the pages in each VM from set $\mathcal{V}^H$ are checked against the pages of all other VMs in $\mathcal{V}^H$ (Lines 15 through 18), while in the second phase the pages of each VM from $\mathcal{V}$ are checked against the active pages on server resource Ω (Lines 32 through 35). Every time a new VM V_j is inserted into $\mathcal{V}^H$, a new efficiency value is calculated (Lines 36 and 37) for every $k > 0$. A test is then performed to recalculate the server capacity reduced by number of pages, K_j, less the shared pages, $\mathcal{S}_j^k$, in common with the active pages on the server resource Ω.

If, by allocating VM V_j onto Ω, the capacity is exceeded, V_j is removed from the "offline" set $\mathcal{V}$ with no opportunity for inclusion in $\mathcal{V}^H$ (Lines 39 through 41). Else, VM V_j is allocated, the server capacity is reduced, and both $\mathcal{V}$ and $\mathcal{V}^H$ are updated accordingly (Lines 42 through 45). Next, pages within the library Π are updated to active, if they have not been already, relative to VM V_j (Lines 46 through 48) and the iterator k is updated (Line 49). Lastly, upon exiting the while loop, server Ω is allocated the subset $\mathcal{V}^H$ of VMs which represents the solution to the SAVMM problem (Line 50).

In the following, we present an example to show how G-SAVMM works. We consider a server with memory capacity $C = 10$ pages. There are twelve distinct pages in the library Π and four VM candidates for allocation onto the server. Figure 1 along with Table II show the details of each iteration k of G-SAVMM. The first column in both Figure 1 and Table II corresponds to the pre-processing phase, where a scan occurs for identical, requested pages within the set of VMs $\mathcal{V}$. In Figure 1, page π_i^j, ($i = 1, \ldots, 12$ and $j = 1, \ldots, 4$), is identified by a block labeled by 1, if it is requested, and by 0, otherwise. The aggregate value of blocks per VM corresponds to the total number of requested pages K_j. The highlighted blocks in Figure 1, correspond to identical pages found between the set of VMs, where $A_i > 1$. The maximum value in A corresponds to the page that is shared the most among all the pages in $\mathcal{V}$. The efficiency metric value is calculated for those VMs sharing this most shared page (i.e., the page with the greatest A_i). Based on the values given in Table II, the highest efficiency metric, 4.772, is associated with V_4, and V_4 is selected for allocation to subset $\mathcal{V}^H$.

The next iteration of G-SAVMM, corresponding to the first iteration of the greedy phase ($k = 1$), is illustrated in the second column of both Figure 1 and Table II. In this iteration a scan occurs for identical, requested pages between VMs and the active pages within library Π. Once the initial VM has been selected for allocation based on the efficiency metric, the provider activates all pages within Π requested by the selected VM. The active pages are identified by blocks with diagonal line filling underneath each page π_i from Π. The active pages correspond to all pages from V_4. The highlighted blocks for VMs in iteration $k = 1$, correspond to those pages that are identical to the active pages in Π. Even though V_1 does not share any active pages with the active pages in Π at $k = 1$, the efficiency metric is calculated and V_1 may be considered a candidate for allocation since at some $k > 1$, there may be active pages that are identical to pages in V_1 in later allocations. The largest efficiency value is 3.250, which corresponds to V_2, and the new server capacity is 6. VM V_2 consists of six pages, where three of them are shared with the active pages in Π and therefore do not have to be accounted for against the capacity. G-SAVMM proceeds until $k = 3$, where the remaining capacity is 1. The total profit obtained

	$k=0$				$k=1$				$k=2$				$k=3$			
	p_j	K_j	$\mathcal{S}_j^0$	$\mathcal{E}_j^0$	p_j	K_j	$\mathcal{S}_j^1$	$\mathcal{E}_j^1$	p_j	K_j	$\mathcal{S}_j^2$	$\mathcal{E}_j^2$	p_j	K_j	$\mathcal{S}_j^3$	$\mathcal{E}_j^3$
V_1	–	–	–	–	6.00	3	0	3.000	6.00	3	1	3.464	6.00	3	1	3.464
V_2	6.50	5	3	3.753	6.50	5	2	**3.250**	–	–	–	–	–	–	–	–
V_3	7.00	5	2	3.500	7.00	5	1	3.131	7.00	5	2	**3.500**	–	–	–	–
V_4	6.75	3	2	**4.772**	–	–	–	–	–	–	–	–	–	–	–	–

Table II: G-SAVMM Execution Example: Efficiency Metric Calculation.

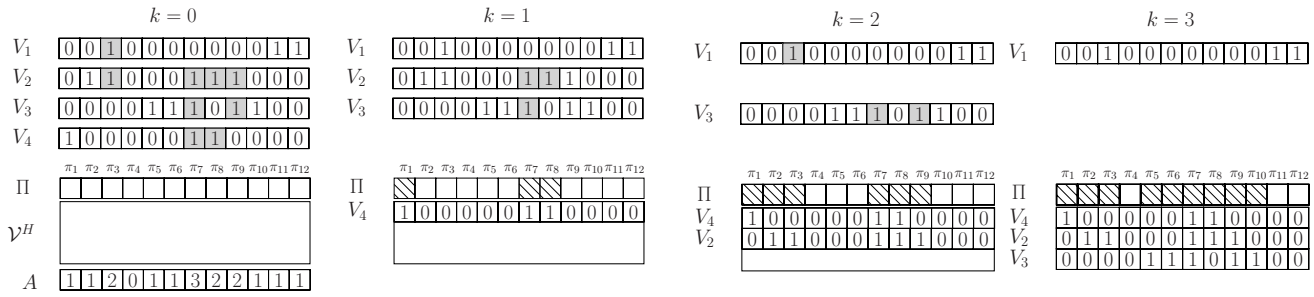

Figure 1: G-SAVMM Execution Example.

by G-SAVMM is 20.25.

IV. G-SAVMM Properties

In this section, we determine the approximation ratio of G-SAVMM and characterize its computational complexity. To develop insight into the properties of G-SAVMM, we design and analyze a worst-case VM instance as follows. Let $\mathcal{V}^W$ denote an instance of the SAVMM problem where VM $V_{\hat{j}}$ does not share any memory pages with the other VMs in $\mathcal{V}^W$. Then, let at least one VM $V_{\hat{j}^c} \in \mathcal{V}^W$ be comprised of pages which are a complement set of pages to VM $V_{\hat{j}}$. In addition, let the remaining VMs in $\mathcal{V}^W$ be comprised of either a subset of pages in VM $V_{\hat{j}^c}$ or be equivalent to VM $V_{\hat{j}^c}$. In either case, the remaining VMs would be allocated onto Ω if $V_{\hat{j}^c}$ were to be allocated first since they all share the same memory pages and would not reduce capacity.

We investigate this instance on a server Ω with capacity C such that either VM $V_{\hat{j}}$ or VM $V_{\hat{j}^c}$ can be allocated, but not both. If VM $V_{\hat{j}^c}$ is allocated, then all remaining VMs in $\mathcal{V}^W \setminus \{V_{\hat{j}}\}$, will be allocated as well due to page sharing. Else, VM $V_{\hat{j}}$ is allocated and utilizes the server resource capacity enough to not allow any other VM to be allocated from $\mathcal{V}^W$. Our last consideration of the problem instance $\mathcal{V}^W$ corresponds to profit. G-SAVMM is inherently sensitive to profit values when calculating the efficiency metric. In the following theorem, we determine the approximation ratio for G-SAVMM based on the worst case instance $\mathcal{V}^W$.

Theorem IV.1. *The approximation ratio of G-SAVMM is M.*

Proof: Let the profit obtained from an optimal solution be denoted as P^* and the optimal set of VMs which generate P^* from $\mathcal{V}^W$ be denoted by $\mathcal{V}_{OPT}^W$, $\mathcal{V}_{OPT}^W \subset \mathcal{V}^W$, where $P^* = \sum_{j:V_j \in \mathcal{V}_{OPT}^W} p_j$. Let the profit obtained by G-SAVMM be denoted by P, and the set of VMs which generate P from $\mathcal{V}^W$ be denoted by $\mathcal{V}_{GRD}^W$, $\mathcal{V}_{GRD}^W \subset \mathcal{V}^W$, where $P = \sum_{j:V_j \in \mathcal{V}_{GRD}^W} p_j$. Assume at $k=0$, VM $V_{\hat{j}}$ is allocated onto Ω; admitting the relationship $\mathcal{E}_j^0 < \mathcal{E}_{\hat{j}}^0$. Then, by Equation 1, $\frac{p_j}{\sqrt{K_j - S_j^0 + 1}} < \frac{p_{\hat{j}}}{\sqrt{K_{\hat{j}} - S_{\hat{j}}^0 + 1}}$. Since VM $V_{\hat{j}}$ does not share pages with VMs in $\mathcal{V}^W$, $\mathcal{S}_{\hat{j}}^0 = 0$, resulting in $\frac{p_j}{\sqrt{K_j - S_j^0 + 1}} < \frac{p_{\hat{j}}}{\sqrt{K_{\hat{j}} + 1}}$, where

$$\frac{\sqrt{K_{\hat{j}} + 1}}{\sqrt{K_j - S_j^0 + 1}} p_j < p_{\hat{j}} \tag{2}$$

establishes the lower bound for $p_{\hat{j}}$ selected according to our efficiency metric at $k=0$. This implies that for any $p_{\hat{j}}$ greater than the established lower bound, VM $V_{\hat{j}}$ will be allocated first onto Ω from $\mathcal{V}^W$ by G-SAVMM. Considering the server utilization of VM $V_{\hat{j}}$ and capacity C, no other VM allocations can be performed and k stops at 0. Since $P = \sum_{j:V_j \in \mathcal{V}_{GRD}^W} p_j$, therefore $P = p_{\hat{j}}$.

Suppose that through an exhaustive search, the optimal value P^*, is calculated whereby VM $V_{\hat{j}^c}$ is allocated first onto Ω at $k=0$. Since every remaining VM in $\mathcal{V}^W$ is comprised of a subset of pages in VM $V_{\hat{j}^c}$, not including VM $V_{\hat{j}}$, then the exhaustive search allocates all remaining VMs onto Ω from $k=1$ to at most $k = M-1$. Thus, the optimal value $P^* = \sum_{j:V_j \in \mathcal{V}_{OPT}^W} p_j$ implies $P^* = \sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} p_j$.

In order to determine the approximation ratio for this instance of SAVMM, we show that $P^* \leq P\alpha$, where α is the multiplicative factor that will give the approximation ratio

of G-SAVMM. Therefore,

$$\frac{P^*}{P} = \frac{\sum_{j:V_j \in \mathcal{V}^W_{OPT}} p_j}{\sum_{j:V_j \in \mathcal{V}^W_{GRD}} p_j} \tag{3}$$

$$= \frac{\sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} p_j}{p_{\hat{j}}} \tag{4}$$

By substituting p_j from Eq. 2, we further determine

$$\frac{P^*}{P} < \frac{\sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} \frac{\sqrt{K_j - S^k_j + 1}}{\sqrt{K_{\hat{j}}+1}} p_{\hat{j}}}{p_{\hat{j}}} \tag{5}$$

$$= \sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} \frac{\sqrt{K_j - S^k_j + 1}}{\sqrt{K_{\hat{j}}+1}} \tag{6}$$

$$= \frac{1}{\sqrt{K_{\hat{j}}+1}} \sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} \sqrt{K_j - S^k_j + 1} \tag{7}$$

For $k > 0$ and $\forall$ VM $V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}$, S^k_j will be at least 1 when VM $V_{\hat{j}^c}$ is allocated first onto Ω. Every remaining VM in $\mathcal{V}^W \setminus \{V_{\hat{j}}\}$, will be allocated onto Ω, where the remaining VMs may only consist of a single shared page with $V_{\hat{j}^c}$ in the worst case. Then,

$$\frac{P^*}{P} \leq \frac{1}{\sqrt{K_{\hat{j}}+1}} \sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} \sqrt{K_j - 1 + 1} \tag{8}$$

$$= \frac{1}{\sqrt{K_{\hat{j}}+1}} \sum_{j:V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}} \sqrt{K_j} \tag{9}$$

Following the allocation of VM $V_{\hat{j}^c}$, we consider $M-1$ maximum number of VMs left to allocate in the optimal solution. Since VM $V_{\hat{j}^c}$ exists and is the complement page set to VM $V_{\hat{j}}$, then for N pages, $1 \leq K_{\hat{j}} \leq N-1$. In addition, since there exists at least 1 shared page index between Λ_j and $\Lambda_{\hat{j}^c}$ $\forall j : V_j \in \mathcal{V}^W \setminus \{V_{\hat{j}}\}$, then for $K_j = 1$ we have

$$\frac{P^*}{P} \leq \frac{(M-1)\sqrt{1}}{\sqrt{K_{\hat{j}}+1}} \tag{10}$$

$$= \frac{M-1}{\sqrt{2}} \tag{11}$$

$$\leq M - 1 < M \tag{12}$$

Therefore, $\frac{P^*}{P}$ is bounded by $\alpha = M$, which results in an approximation ratio of M for the G-SAVMM algorithm. ■

We now investigate the time complexity of G-SAVMM. The running time is dominated by the second phase, the greedy phase. The while-loop (Line 30) may execute a maximum of $M-1$ iterations since one VM has already been inserted into $\mathcal{V}^H$ and there exists instances where $\mathcal{V}^H \subseteq \mathcal{V}$. Within the while-loop, the running time is dominated by the search and calculation of shared pages between the VMs in $\mathcal{V}$ and the active pages on Ω (Lines 32 through 35). The search and calculation are executed a maximum of $M-1$ times, corresponding to the possible number of VMs at $k=1$, by the number of active pages to search on Ω, thus the running time is $O(N(M-1))$. Then, the running time for the entire greedy phase is $O(N(M-1)^2)$. Thus, G-SAVMM has an asymptotic running time of $O(NM^2)$ which is linear in the total number of pages and quadratic in the total number of VMs in the set of "offline" VMs.

V. Experimental Results.

In this section, we perform extensive experiments investigating the performance of G-SAVMM against other VM allocation algorithms considering their obtained profit and the utilization of the server's memory.

A. Experimental Setup

We perform our experiments 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 Version 4.9.0. We compare G-SAVMM with two other VM allocation algorithms: (i) Highest Profit (HP-Oblivious); and, (ii) Maximum Shared Pages (MS-Sharing). The first allocation algorithm, HP-Oblivious, is a greedy algorithm which allocates VMs in decreasing order of their profit and is page sharing oblivious. The second allocation algorithm, MS-Sharing, is a greedy algorithm which allocates VMs in decreasing order of their number of shared pages. The page sharing consideration in MS-Sharing mirrors that of G-SAVMM, but it does not take into account the profit.

Our environment assumes page sharing within each simulation we evaluate. We consider the degree of sharing among the VMs and categorize the SAVMM instances into four categories, called *sharing stratifications*: (i) *Low-Share* (no greater than 20% of the active pages on the server are shared with VMs); (ii) *Mid-Share* (no greater than 50% of the active pages on the server are shared with VMs); (iii) *High-Share* (no greater than 80% of the active pages on the server are shared with VMs); and, (iv) *Full-Share* (approx. all active pages on the server are shared with VMs). Our experiments consist of 1000 simulations per sharing stratification. In our simulations the degree of sharing in each sharing stratification are within the following ranges: (i) 15%−20% for Low-Share; (ii) 38%−50% for Mid-Share; (iii) 70%−80% for High-Share; and, (iv) 92%−99% for Full-Share.

Each instance of SAVMM considered in the simulation consists of 10 VMs. Each VM is assigned a profit value randomly, ranging from \$1 to \$20. The number of pages is also generated randomly with a maximum of 1000 pages possible per VM. Our server capacity C is fixed at 60% of the total number of pages for each simulation. Based on our experiments, operating at 60% capacity provides enough resources to accommodate a wide variety of simulations.

Our criterion for identifying the best performing algorithm is based on profit ratios. In our experiments, we execute the three greedy algorithms HP-Oblivious, MS-Sharing and G-SAVMM on various instances of the SAVMM problem. The profit of the VMs generated for each of the SAVMM instances considered in the experiments varies within the range specified in the previous paragraph. Thus, comparing and

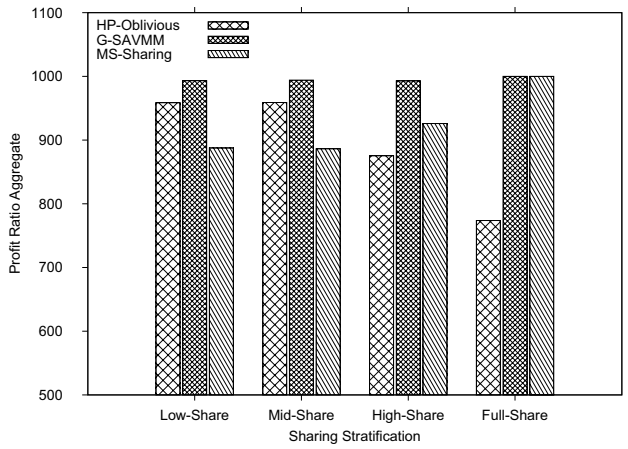

Figure 2: Profit Ratios vs. Sharing Stratifications.

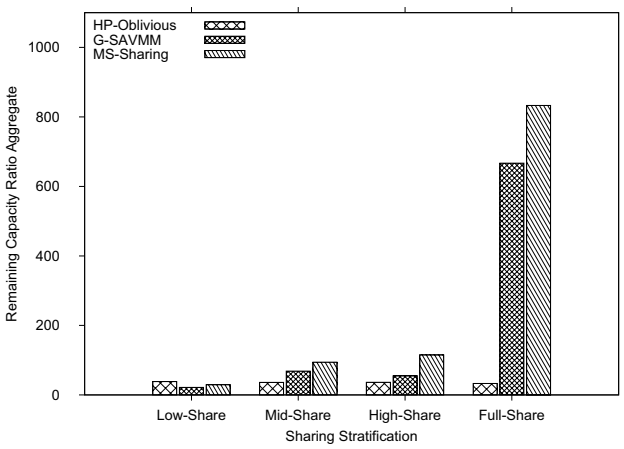

Figure 3: Capacity Ratios vs. Sharing Stratifications.

then aggregating the actual values of the profit generated by each of these greedy algorithms over a number of simulations provides misleading information on the performance of the algorithms. Instead, we compare the profits generated by each greedy algorithm over the maximum profit generated for an instance and aggregate those ratios for a specific number of instances. For example, suppose after simulating an instance of the SAVMM problem, HP-Oblivious generates a profit value of 100, MS-Sharing generates a profit value of 200 and G-SAVMM generates a profit value of 250. Then, the maximum profit generated for that instance would be 250. The calculated profit ratios would be .4, or $\frac{100}{250}$, for HP-Oblivious, .8, or $\frac{200}{250}$, for MS-Sharing and 1, or $\frac{250}{250}$, for G-SAVMM. The profit ratios indicate each greedy algorithm's proximity to the maximum profit attained for that instance. These profit ratios will never be larger than 1 for any of the algorithms and any instances. By aggregating these ratios over 1000 simulations, we identify the best performing algorithm as the one with the highest profit ratio aggregate. The profit ratio aggregate for each algorithm over the course of 1000 simulations will never be larger than 1000. In addition, these 1000 simulations are performed for each sharing stratification to determine the best performing algorithm under the various sharing scenarios.

B. Analysis of Results

We now compare the performance of G-SAVMM against both HP-Oblivious and MS-Sharing algorithms. In Figure 2, we plot the aggregate profit ratios of all three algorithms under different sharing stratifications. For sharing stratifications Low-Share, Mid-Share and High-Share, G-SAVMM outperforms both HP-Oblivious and MS-Sharing algorithms. In Low-Share, G-SAVMM resulted in either the profit maximum over or equal to the profits obtained using HP-Oblivious and MS-Sharing, in 852 of the 1000 simulations. In Mid-Share, G-SAVMM resulted in either the profit maximum over or equal to the profits obtained using HP-Oblivious and MS-Sharing in 875 of the 1000 simulations. In High-Share, G-SAVMM resulted in either the profit maximum over or equal to the profits obtained using HP-Oblivious and MS-Sharing in 816 of the 1000 simulations. In the Low-Share and Mid-Share stratifications, our experiments have shown that HP-Oblivious outperforms MS-Sharing. In the High-Share and Full-Share stratifications, our experiments have shown that MS-Sharing outperforms HP-Oblivious. As the sharing potential in the stratification increases, MS-Sharing generates an increased profit since more VMs may be allocated. In the Full-Share stratification, G-SAVMM and MS-Sharing generate the same profit resulting in a profit maximum in 1000 out of 1000 simulations. Based on our results, G-SAVMM attains a profit ratio aggregate of: (i) 993.2759 for Low-Share; (ii) 994.0514 for Mid-Share; (iii) 992.9242 for High-Share; and, (iv) 1000 for Full-Share. When a simulation contains VMs with full-sharing potential, G-SAVMM or MS-Sharing returns the same result. When the simulated instance consists of VMs with less opportunity to share pages, G-SAVMM is the preferred algorithm with respect to profit maximization. Therefore, according to our experiments, G-SAVMM should be the chosen algorithm for solving SAVMM.

In Figure 3, we plot the aggregate remaining memory capacity ratios, after the VMs have been allocated, for all three algorithms under different sharing stratifications. We have shown the efficacy of G-SAVMM for profit maximization now we show that from the point of view of preserving resources, G-SAVMM also performs well. The remaining capacities are slightly larger for HP-Oblivious in the Low-Share and are larger for MS-Sharing in Mid-Share and High-Share. The significant differences between these algorithms occur in the Full-Share stratification. MS-Sharing dominates the amount of unused capacity with G-SAVMM also experiencing a higher unused capacity; albeit not as significant as MS-Sharing, yet well above HP-Oblivious. Therefore, choosing G-SAVMM as the algorithm for solving SAVMM leads to a considerable saving of memory which can be utilized for other purposes.

VI. Conclusion

We designed a sharing-aware greedy approximation algorithm (G-SAVMM) for solving the sharing-aware VM maximization problem. We showed that G-SAVMM is a M-approximation algorithm, where M is the number of VM instances that need to be allocated. The experimental results show that G-SAVMM outperforms two other VM allocation algorithms in terms of generated profit.

In future work, we plan on extending G-SAVMM to manage the VM allocation process in online environments. In addition, we plan on extending G-SAVMM considering multi-criteria objectives such as energy and profit.

Acknowledgment

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

References

[1] VMWare. (2006) Virtualization overview. [Online]. Available: http://www.vmware.com/pdf/virtualization.pdf

[2] M. Jeyakanthan and A. Nayak, "Policy management: leveraging the open virtualization format with contract and solution models," *IEEE Network*, vol. 26, no. 5, pp. 22–27, September 2012.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[4] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 9, pp. 34–45, Sep. 1974.

[5] C. A. Waldspurger, "Memory resource management in vmware esx server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, Dec. 2002.

[6] M. Sindelar, R. Sitaraman, and P. Shenoy, "Sharing-aware algorithms for virtual machine colocation," in *Proc. of the 23rd ACM symposium on Parallelism in Algorithms and Architectures*, New York, NY, USA, 2011, pp. 367–378.

[7] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2004.

[8] T.-C. Lai, "Worst-case analysis of greedy algorithms for the unbounded knapsack, subset-sum and partition problems," *Oper. Res. Lett.*, vol. 14, no. 4, pp. 215–220, Nov. 1993.

[9] M. I. Sviridenko, "Worst-case analysis of the greedy algorithm for a generalization of the maximum p-facility location problem," *Oper. Res. Lett.*, vol. 26, no. 4, pp. 193–197, May 2000.

[10] S. Zaman and D. Grosu, "Combinatorial auction-based mechanisms for vm provisioning and allocation in clouds," in *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2012, pp. 729–734.

[11] M. M. Nejad, L. Mashayekhy, and D. Grosu, "A family of truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," in *Proc. of IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 188–195.

[12] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Almost optimal virtual machine placement for traffic intense data centers," in *INFOCOM*, 2013, pp. 355–359.

[13] S. Takahashi, A. Takefusa, M. Shigeno, H. Nakada, T. Kudoh, and A. Yoshise, "Virtual machine packing algorithms for lower power consumption," in *Proc. of the IEEE 4th International Conference on Cloud Computing Technology and Science*, Dec 2012, pp. 161–168.

[14] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum, "Disco: Running commodity operating systems on scalable multiprocessors," *ACM Trans. Comput. Syst.*, vol. 15, no. 4, pp. 412–447, Nov. 1997.

[15] I. Banerjee, P. Moltmann, K. Tati, and R. Venkatasubramanian. (2013) Esx memory resource management: Transparent page sharing. [Online]. Available: https://labs.vmware.com/academic/publications/vmware-white-papers

[16] 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. of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2009, pp. 31–40.

[17] X. Project. (2010) Xen 4.0 release notes. [Online]. Available: http://wiki.xen.org/wiki/Xen_4.0_Release_Notes

[18] D. Gupta, S. Lee, M. Vrable, 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.