University of Massachusetts Amherst

From the SelectedWorks of Ramesh Sitaraman

June, 2011

Sharing-aware Algorithms for Virtual Machine Colocation

Michael Sindelar
Ramesh Sitaraman, *University of Massachusetts - Amherst*
Prashant Shenoy

Available at: https://works.bepress.com/ramesh_sitaraman/22/

Sharing-Aware Algorithms for Virtual Machine Colocation

Michael Sindelar
Google Inc.
1600, Amphitheatre Parkway
Mountain View, CA 94043
sindelar@google.com

Ramesh K. Sitaraman
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
ramesh@cs.umass.edu

Prashant Shenoy
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
shenoy@cs.umass.edu

ABSTRACT

Virtualization technology enables multiple virtual machines (VMs) to run on a single physical server. VMs that run on the same physical server can share memory pages that have identical content, thereby reducing the overall memory requirements on the server. We develop sharing-aware algorithms that can colocate VMs with similar page content on the same physical server to optimize the benefits of inter-VM sharing. We show that inter-VM sharing occurs in a largely hierarchical fashion, where the sharing can be attributed to VM's running the same OS platform, OS version, software libraries, or applications. We propose two hierarchical sharing models: a tree model and a more general cluster-tree model. Using a set of VM traces, we show that up to 67% percent of the inter-VM sharing is captured by the tree model and up to 82% is captured by the cluster-tree model. Next, we study two problem variants of critical interest to a virtualization service provider: the VM Maximization problem that determines the most profitable subset of the VMs that can be packed into the given set of servers, and the VM packing problem that determines the smallest set of servers that can accommodate a set of VMs. While both variants are NP-hard, we show that both admit provably good approximation schemes in the hierarchical sharing models. We show that VM maximization for the tree and cluster-tree models can be approximated in polytime to within a $(1 - \frac{1}{e})$ factor of optimal. Further, we show that VM packing can be approximated in polytime to within a factor of $O(\log n)$ of optimal for cluster-trees and to within a factor of 3 of optimal for trees, where n is the number of VMs. Finally, we evaluate our VM packing algorithm for the tree sharing model on real-world VM traces and show that our algorithm can exploit most of the available inter-VM sharing to achieve a 32% to 50% reduction in servers and a 25% to 57% reduction in memory footprint compared to sharing-oblivious algorithms.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SPAA'11, June 4–6, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0743-7/11/06 ...$10.00.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; D.4.2 [**Operating Systems**]: Storage Management—*Main memory*; D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*

General Terms

Algorithms, Management, Measurement, Theory

Keywords

Virtualization, Optimization, Page Sharing, Bin Packing

1. INTRODUCTION

Modern data centers that incorporate large server farms increasingly employ a virtualized architecture where applications run inside virtual machines (VMs) that are then hosted on physical servers. An ever-increasing range of applications utilize data center virtualization, including web hosting, enterprise applications, and e-commerce. The goal of a data center service provider, such as a cloud computing provider, is to maximize the utility that the VMs provide while conserving the required server resources such as memory and CPU.

Virtualization technology enables multiple VMs to run on the same physical server. VMs running on the same physical server share resources such as CPU and memory by utilizing a key component called the *hypervisor*. To enable the conservation of memory resources, modern hypervisors such as VMware ESX support content-based sharing of memory pages [14]. Specifically, if multiple VMs resident on the same physical server use identical pages, content-based sharing allows storing just one copy of the shared page. Thus, content-based sharing decreases the memory footprint required to host a set of VMs on a single physical server. The concept of content-based sharing was first utilized in the Disco system [1], and subsequently implemented in VMware ESX [14] where the technique was shown to save as much as 33% of the memory resources of a server. However, content-based sharing is effective only if it is complemented by algorithms that ensure that the VMs resident on each physical server contain a significant amount of sharable pages. Specifically, to truly utilize the potential of inter-VM page sharing and to significantly reduce the aggregate memory footprint, it is essential that VMs with the most shared pages are co-located in the same physical server. This underscores the importance of "sharing-aware" algorithms that "pack" VMs

into servers in a manner that VM page sharing is maximized and the total memory footprint is minimized. The potential for exploiting inter-VM sharing by intelligent colocation was recognized in [16] where the authors developed a fingerprinting scheme that provides a compact representation of the memory pages in a VM. Such a scheme is an essential step in identifying VMs with a large page sharing potential. In our paper, we go further by developing formal models and algorithms for sharing-aware placement of VMs on physical servers. While we explicitly model only memory resources, other resources such as CPU and network are additional considerations in VM placement. Extending the results and techniques in this paper to multiple resources is an important direction for future work.

1.1 Our contributions

Our first contribution is the design of graph models to capture page sharing across a set of virtual machines and to empirically demonstrate the efficacy of our models to capture sharing in real systems. Our graph models of sharing form the basis of our algorithmic study and are also likely critical for future algorithmic studies in the area. We present a general sharing model and two variants of hierarchical sharing, namely the tree and the cluster-tree models. Our hierarchical models assume that shared pages between VMs can be attributed to commonality in a hierarchy of dimensions such as the OS platform, OS version, software libraries, and types of applications. Using memory traces for a mixture of diverse OSes, architectures, and software libraries, we find that a tree model can capture up to 67% of inter-VM sharing from these traces, whereas the more general cluster-tree model can capture up to 82% of the inter-VM sharing. These results demonstrate the utility of our models in capturing real-world memory sharing.

Our second contribution is the formulation and development of sharing-aware algorithms for two optimization problems that are key to a virtualization service provider: the *VM maximization* problem and the *VM packing* problem. We study these problems in the general sharing model as well as in the two hierarchical sharing models. Interestingly, we find that although the hierarchical sharing models ignore some types of sharing, thereby reducing the sharing potential, they significantly help the design of provably-good approximation algorithms for these problems. Our theoretical results are summarized in Figure 1.

Sharing Model	VM Maximization	VM Packing
General	No approx within $2^{(\log n)^{\delta}}$	Open
Cluster-Tree (Hierarchical)	FPTAS	$O(\log n)$-approx
Tree (Hierarchical)	FPTAS	3-approx

Figure 1: Summary of theoretical results

In the *VM maximization problem*, we are given a set of n VMs, where each VM is associated with a profit value that is earned by hosting that VM. Each VM consists of a set of memory pages. We are also given m servers, each server with a capacity of P pages. The goal of VM maximization is to determine the subset of the VMs that can be hosted on the m servers so that the total profit earned is maximized. This problem characterizes the desire of a service provider to maximize the earned profit for a fixed set of server resources. We show that VM maximization is NP-Hard and is infeasible to even approximate well in the general sharing model where each VM can contain an arbitrary set of pages. Specifically, we show that it is infeasible to derive an approximate solution that is within a factor of $2^{(\log n)^{\delta}}$ of optimal, for some $\delta > 0$, assuming that 3-SAT $\notin DTIME(2^{n^{3/4+\epsilon}})$. However, we show that in the cluster-tree model and the special case of a single server, we can devise a fully-polynomial time approximation scheme (FPTAS) that yields an approximate solution that is at least a factor of $(1-\epsilon)$ of the optimal profit, for any $\epsilon > 0$. Further, this result can be extended to obtain an approximate solution that is at least a factor of $(1-\frac{1}{e})$ of the optimal profit for the VM maximization problem with multiple servers.

In the *VM packing problem* problem, we are given a set of VMs that must be hosted on physical servers, where each VM consists of a set of pages. The goal of this problem is to "pack" the VMs onto the smallest number of physical servers, where each server has a capacity of P pages. While VM packing is also NP-Hard, we show that we can efficiently compute an approximate solution that is within a factor of $O(\log n)$ of optimal for cluster trees and a factor 3 of optimal for trees, where n is the number of VMs. From a theoretical perspective, the VM packing problem is an interesting new generalization of the well-studied bin packing problem. Unlike bin packing, in VM packing the cumulative size of a set of items (i.e., VMs) in a bin (i.e., server) can be *smaller* than the sum of individual item sizes due to sharing. Note that if sharing is ignored, VM packing reduces to traditional bin packing.

The VM packing problem fundamentally characterizes the goals of a service provider to host a set of VMs using the least amount of server resources. The service provider will need to periodically allocate VMs to servers in a sharing-aware fashion, either as a part of the initial placement of newly-created VMs or as part of a "repacking" operation of existing VMs [16]. A periodic repacking would be necessary as both the pool of existing VMs and the sharing characteristics of individual VMs change over time, making the initial packing potentially suboptimal.

Our third contribution is an experimental evaluation of our VM packing algorithm for the tree model of sharing on actual end-user VM traces. We find that our sharing-aware algorithm reduces the number of servers required by 32% to 50% when compared to a sharing-oblivious Modified First Fit Decreasing (MFFD) bin packing algorithm. Our algorithm also significantly reduced the memory usage with the total memory footprint across all servers decreasing by 25% to 57% when compared with any sharing-oblivious algorithm. We further show that loading each server to at most 90% of memory capacity produces colocations that are stable with respect to the fluctuations in the VM's memory usage and inter-VM sharing over time.

2. VM MEMORY SHARING: PROPERTIES AND MODELS

We present three models that capture inter-VM memory page sharing with different levels of complexity. We then use real VM traces to show that a significant portion of the inter-

VM page sharing can be captured by simpler structured hierarchical models that facilitate the design of provably good sharing-aware VM colocation algorithms.

2.1 Graph models for page sharing

Our first model is the general sharing model that can accurately capture all inter-VM sharing. In this model, each VM consists of an arbitrary set of pages. One can view this as a hypergraph $G = \langle \mathrm{V}, E \rangle$, where V is the set of VMs and each hyperedge $e \subseteq \mathrm{E}$ denotes a memory page that is shared by all the VMs in the hyperedge. Clearly, the general sharing model can capture arbitrary sharing of pages between the VMs.

In the two hierarchical sharing models that we propose, sharing cannot be arbitrary and only some forms of sharing can be captured. First, we propose a tree model where sharing is modeled as a rooted directed tree $\mathrm{T} = \langle V, E \rangle$ where the edges are directed away from the root and towards the leaves (See Figure 2). Each node in $v \in V$ is associated with $w(v)$ distinct pages. Each leaf corresponds to a VM and pages that are unique to that VM are associated with it. The pages associated with a non-leaf node v is shared by all the VMs that correspond to leaves of the subtree of T rooted at node v. Thus, the set of all pages in a VM is the union of all pages associated with nodes on the corresponding root-to-leaf path in T.

An example of a tree sharing model is shown in Figure 2 where one can view each level of bifurcation as representing dimensions such as OS, OS version, or software libraries. Page sharing between any two VM's is attributed to commonality in those dimensions. For instance, the pages shared across all VMs is captured in the root in the first level of the tree. VMs running the same OS platform may share a set of pages related to that OS and is captured in the second level of the tree. If they both also run the same OS version there is additional sharing captured at the third level of the tree. Even more sharing occurs if they also use the same software libraries that is captured in the fourth level of the tree. Finally, pages that are unique to the VM is captured in the corresponding leaves of the tree.

A more general version of the hierarchical sharing model is the cluster-tree model whose nodes and edges are "clustered" into super-nodes and super-edges respectively, and the super-nodes and super-edges form a rooted directed tree (See Figure 3). More formally, a cluster-tree consists of a rooted directed tree $\mathrm{T} = \langle \mathrm{V}, \mathrm{E} \rangle$, where each $\mathrm{v} \in \mathrm{V}$ is a *super-node* and each $(\mathrm{u}, \mathrm{v}) \in \mathrm{E}$ is a *super-edge*. Each super-node $\mathrm{v} \in \mathrm{V}$ contains a distinct set of one or more nodes denoted by $\Gamma(\mathrm{v})$ such that for $\mathrm{u} \neq \mathrm{v}$, $\Gamma(\mathrm{u}) \cap \Gamma(\mathrm{v}) = \emptyset$. Each super-edge $(\mathrm{u}, \mathrm{v}) \in \mathrm{E}$ contains a distinct set of one or more directed edges denoted by $\Gamma(\mathrm{u}, \mathrm{v})$ where $\Gamma(\mathrm{u}, \mathrm{v}) \subseteq \Gamma(\mathrm{u}) \times \Gamma(\mathrm{v})$. The *width* of a cluster-tree denoted by k equals the $\max_{\mathrm{v} \in \mathrm{V}} |\Gamma(\mathrm{v})|$ and is assumed to be a constant. Note that setting k equal to 1 reduces a cluster-tree to the simpler tree model. Therefore, any algorithmic results for the cluster-tree model also apply to the tree model.

A cluster-tree can be used to model a more complex form of inter-VM sharing as follows. As before, each node v is associated with $w(v)$ distinct pages. Each super-node that is a leaf of T corresponds to a VM and contains a single node. Let a VM correspond to a leaf super-node that contains a single node v. All *unique* pages of that VM are associated with node v. Further, the set of *all* pages in that VM is the union of all pages associated with nodes u such that either $u = v$ or u has a (directed) path to v. For instance, the leftmost VM in the cluster-tree of Figure 3 contains the unique pages associated with its leaf node, the pages associated with the "mail" and "browser" nodes that are shared with other VMs that use those desktop applications, and the pages associated with the root node that are shared with all VMs in the cluster-tree.

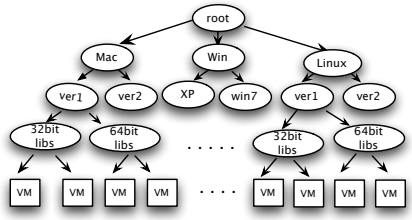


Figure 2: Tree model example

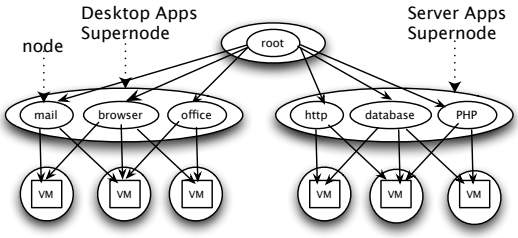


Figure 3: Cluster-Tree model example

The cluster-tree model captures page sharing more accurately than the tree model in certain situations. As a hypothetical example, a set of VMs might be using desktop applications, while another set of VMs might be using server applications (See Figure 3). However, each VM may use only a proper subset of these applications. To capture this situation one can use a single super-node to represent desktop applications, and another super-node to represent server applications. The super-nodes contain individual nodes representing each individual application. We can then connect each leaf representing a VM to the proper subset of the nodes representing applications that the VM uses, thereby modeling the sharing more accurately. For instance, in Figure 3, the leftmost VM only uses a subset of the desktop applications, namely "mail" and "browser' but not "office". We have found that a cluster-tree with small constant width is typically sufficient for effectively capturing additional inter-VM sharing.

2.2 Empirical analysis of inter-VM sharing

Both the tree and cluster-tree hierarchical models are more structured than the general model, but they typically do not capture all of the inter-VM page sharing that exists in a set of VMs. However, using VM memory traces, we show that in practice these two models are able to capture a majority of the inter-VM sharing. The intuitive reason for their efficacy is because the structure of these models reflects how VM sharing occurs in practice. Typically the amount of memory pages that are shared between any two VMs depends

on the OS platform, OS version and the software libraries utilized by those VMs. For example, more memory pages are likely to be shared between VMs running the same OS platform (e.g.,between two Mac or two Windows VMs) than between those running different OS platforms (e.g., between a Mac and a Windows VM). Similarly, there is likely to be more sharing between VMs running the same OS versions. For instance, more sharing is likely between two VMs that both run Windows XP and than two VMs that run Windows XP and Windows 7 respectively. Finally, software libraries also govern the amount of likely sharing—similar libraries and library versions will yield more sharing. As depicted in Figures 2 and 3, hierarchical models are well suited to capture such sharing by grouping VMs based on the OS platform, OS version, software library versions and so on. A limitation though is when VMs with the same OS platform and version run different overlapping subsets of application processes or have different overlapping subsets of software libraries installed on them. Sharing of this nature is not easily captured by hierarchical models.

2.2.1 Trace description

To evaluate the models and algorithms in this paper, we use a set of VM memory traces. Our dataset consists of traces from over 50 machines. We gathered memory traces from 31 volunteer machines in our department over a 9 month period. Since these volunteer machines comprised of only a subset of OS platforms and versions that we wanted to study (e.g., Windows machines and newer Linux versions were under-represented in our volunteer dataset), we setup 20 additional virtual machines in our laboratory testbed and also gathered traces from these synthetic machines, giving us the full diversity of OS platforms, OS versions, and software libraries for our study. Table 1 summarizes the various machines in our trace dataset.

OS	CPU	RAM
Mac OSX 10.5	PowerBook 6, PowerPC	1152MB
Mac OSX 10.5	i386	1GB,2GB
Centos Linux 4.4	Intel x86	1GB,2GB
Centos Linux 5.4	Intel x86	256MB
Centos Linux 5.4	Intel x86	1GB,2GB
Centos Linux 5.5	Intel x86	512MB
Centos Linux 5.5	Intel x86	1GB,2GB
Centos Linux 5.5	Intel x86 VMware	1536MB
Mac OSX 10.6	Intel x86 VMware	1536MB
Ubuntu 10.4, 10.10	Intel x86 VMWare	1536MB
Windows XP SP2	Intel x86 VMWare	1536MB
Windows 7	Intel x86 VMWare	1536MB

Table 1: Configuration of machines used to gather memory traces in our study

2.2.2 Properties of inter-VM sharing

We first consider the amount of inter-VM page sharing that is present under the general sharing model. If all virtual machines are allocated to a hypothetical physical server with infinite memory capacity, then approximately 13% of pages can be removed due to inter-VM sharing.[1] As shown in Table 2, the average sharing between pairs of Windows machines was about 8.83%. Between Linux machines, the average was 2.24%. Not surprisingly, the average sharing across OS platforms (i.e., across Windows, Linux, and Mac machines) was at most 0.2%, i.e., it was an order of magnitude smaller. The inter-VM sharing increased significantly when the OS version used by the VMs were also the same. Table 3 shows the sharing observed across OS versions of the same OS platform. Since the trends are similar across OS platforms, only results for some Linux versions are shown here as a representative example. Note that while we observe 10-14% sharing across VMs running the same Linux OS versions in Table 3, the sharing drops to 4.5% for different versions of the same Linux Ubuntu distribution. Finally, we find little sharing across different Linux distributions (e.g., Centos and Ubuntu).

	Linux	Windows	Mac
Linux	2.24%	0.09%	0.02%
Windows		8.83%	0.16%
Mac			3.29%

Table 2: Average sharing between VM pairs across OS platforms.

	CentOs 5.5	Ubuntu 10.4	Ubuntu 10.10
CentOs 5.5	10.56%	0.37%	0.38%
Ubuntu 10.4		13.99%	4.51%
Ubuntu 10.10			13.80%

Table 3: Average sharing between VM pairs across Linux OS versions

2.2.3 Efficacy of the tree and cluster-tree models

To study the efficacy of the tree and cluster-tree models, we selected traces from the 20 virtual machines in our laboratory testbed that comprised our synthetic data set. This provided us with a more controlled sample that represented the full diversity of OS platforms, OS versions, and software libraries. Specifically, our chosen data set represented all three OS platforms: Macs, Windows and Linux and a multitude of OS versions (Ubuntu 10.4 and 10.10, CentOS 5.5, Windows XP and 7, and Mac OS X 10.6); where available, we chose 32- and 64-bit OS versions as separate traces. The VM traces used in this experiment were carefully chosen to represent both server as well as desktop environments. Our desktop VMs comprised libraries and running processes for OpenOffice.org (v. 3.2), Firefox (v. 3), Adobe Flash (v. 10.1), and iTunes (v. 10.1), while our server VMs comprised libraries and processes for Apache 2, PHP 5, and MySQL 5.

We built hierarchical models for our chosen set of VM traces to study how much of the inter-VM sharing can be captured by these models. Our traces lend themselves naturally to a tree model which we consider first (See Figure 2). We constructed a tree with a single root that represents the set of all VMs, that has a child for each OS platform (Mac, Linux, Windows), that each have children for each OS version, which in turn have children for 32- and 64- bit versions of the software libraries. The leaves of the tree are individual VMs that run our desktop and server applications. Starting with the root, at each node v, we computed the

[1]As in [16], our analysis assumes that self-shared pages—duplicated pages within a VM—are removed prior to computing inter-VM sharing. Our traces contained around 9.5% self-shared pages.

	Tree	Cluster-Tree
OS	0.24%	11.37%
OS Ver	11.48%	15.68%
32- v 64-bit libs	55.27%	55.28%
Total	67.01%	82.33%

Table 4: Percentage of sharing captured in the two hierarchical models

pages that are common to all VMs that are descendants of v and associated those pages with node v. Subsequently, these pages were removed from the VMs that are descendants of v and the process was repeated recursively for each child of v. Note that that this process captures some but not all of the inter-VM sharing. Specifically, if a page p is shared between two VMs u and v but p is not present in *all* VMs that are descendants of the lowest common ancestor of u and v then page p is not captured by the tree model of sharing. However, for our VM traces, we found that the tree model captures 67% of the inter-VM sharing when compared to the general model that captures all inter-VM sharing (See Table 4).

The tree decomposition also provides insight into the amount of sharing at different levels—the amount of sharing increases as we descend down the tree. That is, only a small fraction of the sharing is attributable solely to the OS platform; a larger fraction is attributable solely to the same OS version, and a even greater amount to the architecture-dependent software libraries. At the software library level, 55% of the sharing is captured, even across VMs running different sets of applications. Thus, most of the inter-VM sharing comes from pages specific to the OS and software library versions of a VM.

One notable limitation of the tree model constructed above is that it fails to capture sharing due to 64-bit VMs that run both 32-bit and 64-bit software libraries. In the tree model of Figure 2, a VM can either use 32-bit software libraries or 64-bit software-libraries, but not both. Our cluster-tree model can be made to capture such sharing in the following manner. We convert each node of the tree to be a super-node that can contain one or more nodes. Then, we include a special node in each super-node that captures the pages that are shared between all 32-bit software libraries (in our case, 32-bit OpenOffice libraries installed on both 32- and 64-bit VMs). In this fashion, we are able to account for an additional 15% of sharing, giving a total of 82% of potentially sharable pages being captured by our cluster-tree model (See Table 4).

In summary, our hierarchical models are able to capture a significant amount of inter-VM sharing—in our analysis, a simple tree structure could account of 67% of the total sharing, while adding a small amount of complexity enabled a cluster-tree of width 2 to capture 82% of the sharing. As we show next, such hierarchical models also enable design of provably-good approximation algorithms for exploiting inter-VM sharing.

3. ALGORITHMS FOR VM COLOCATION

We study two colocation problems from the standpoint of a virtualization service provider. First, we study the *VM maximization* problem where we are given m servers that can each hold P memory pages, a set of virtual machines $V = \{v_1, v_2, \cdots v_n\}$, and a profit function $p : V \to \mathbb{Z}^+$. Each virtual machine v_i is represented as a set P_i of pages. The goal is to find a set $V' \subseteq V$ such that V' can be packed into the m servers such that the memory capacities of the servers are not exceeded and the profit $\sum_{v_i \in V'} p(v_i)$ is maximized.

We also study the *VM packing problem* where we allocate a set of virtual machines to servers such that total number of servers is minimized, i.e., the hardware resources utilized by the service provider are minimized. Specifically, we are given a set of virtual machines $V = \{v_1, v_2, \cdots v_n\}$, where each virtual machine v_i contains a set of pages P_i. The goal is to allocate all VMs to servers such that the memory capacity P of each server is not exceeded and the total number of servers are minimized.

Both the VM maximization problem and the VM packing problem are NP-Hard, since they contain the knapsack problem and the bin-packing problem respectively as special cases when VMs do not share pages. However, as we shall see, the complexity of finding a provably approximate solution to either colocation problem is crucially dependent on the page sharing model.

3.1 General Sharing

The advantage of the general sharing model where any VM can contain any subset of pages is that all inter-VM sharing can be captured in this model. However, provably-good polytime approximation algorithms for VM colocation may not exist in the general sharing model. For instance, we can show that the VM Maximization is hard to even approximate in the general sharing model.

DEFINITION 3.1.1 (DENSEST k-SUBHYPERGRAPH PROBLEM). *Given a hypergraph $G = (V, E)$ and a parameter k, the* densest k-subhypergraph problem *is to find a set of k vertices with maximum number of hyperedges in the subgraph induced by this set.*

THEOREM 3.1.2 (HAJIAGHAYI ET. AL. [6]). *The densest k-subhypergraph problem is hard to approximate within a factor of $2^{(\log n)^\delta}$ for some $\delta > 0$ under the assumption that 3-SAT $\notin DTIME(2^{n^{3/4+\epsilon}})$.*

THEOREM 3.1.3. *The VM Maximization problem is hard to approximate within a factor of $2^{(\log n)^\delta}$ for some $\delta > 0$ under the assumption that 3-SAT $\notin DTIME(2^{n^{3/4+\epsilon}})$.*

PROOF. Let each page be a vertex in a hypergraph and each VM be a hyperedge that connects each vertex corresponding to a page contained in the VM. Further, let the profit of packing any VM be equal to 1. Then the problem of maximizing the number of VMs allocated to a server of size k is simply finding the densest k-subhypergraph. The result follows from Theorem 3.1.2. □

The complexity of approximating the VM packing problem in case of general sharing is open (See Figure 1).

3.2 Hierarchical sharing

The hierarchical sharing models do not capture all sharing that exists between VMs, but as we concluded in Section 2, it captures the majority of the sharing that exist between VMs. As we show in this section, the advantage of the hierarchical models is that it is more amenable to provably-good polytime approximation algorithms for both VM maximization and VM packing.

3.2.1 VM Maximization

We now show that in the hierarchical sharing model the VM maximization problem admits good approximation algorithms in the form of an FPTAS (fully polynomial time approximation scheme). In this section, we use the more general cluster-tree model of hierarchical sharing. The result also holds for the tree model, since the tree model is a special case of the cluster-tree model.

First, we develop a dynamic programming solution for the simpler version of the problem where we have only one server, using the left-right dynamic programming technique of [7]. We are given a cluster-tree $\mathrm{T} = \langle \mathrm{V}, \mathrm{E} \rangle$ where each leaf of T represents a VM and contains a single node v whose packing yields a profit $p(v) \in \mathbb{Z}^+$. Each non-leaf is super-node consisting of at most k nodes, where k is the width of the cluster-tree. The profit values $p(v)$ are uniformly zero for any node v contained in a non-leaf super-node, i.e., profit is gained only on a leaf node which results in an entire VM being packed. Each node v is either contained in a leaf or is contained in a non-leaf super-node of T. Recall that each node v is associated with a set of $w(v)$ distinct pages. Note that the pages associated with leaves are unique to that VM and the pages associated with nodes contained in non-leaf super-nodes are potentially shared between multiple VMs. Let $\chi[\mathrm{c}, \sigma, j, \beta]$ be the smallest size (number of pages) that must be packed into the single server to achieve a profit of at least β, with the constraint that only the subset σ of nodes contained in super-node c is packed and the remainder of the nodes are selected from trees rooted at the first j child super-nodes of super-node c.

$$\chi[\mathrm{c}, \sigma, 0, \beta] = \begin{cases} \sum_{v \in \sigma} w(v) & \text{if } \sum_{v \in \sigma} p(v) \geq \beta \\ \infty & \text{if } \sum_{v \in \sigma} p(v) < \beta \end{cases}$$

Let c_j be the j^{th} child of c and C' be the set of all subsets σ' of nodes in super-node c_j such that σ' and σ are compatible, i.e, for any node $v \in \sigma'$ all parents of v are included in σ. (Note that compatibility ensures that a node is packed only if its ancestors are also packed.) Thus, $\chi[\mathrm{c}, \sigma, j, \beta]$ equals

$$\min_{0 \leq \beta' \leq \beta} \left\{ \chi\left[\mathrm{c}, \sigma, j-1, \beta - \beta'\right] + \min_{\sigma' \in C'} \chi\left[\mathrm{c}_j, \sigma', degree(\mathrm{c}_j), \beta'\right] \right\},$$

where $degree(\mathrm{c}_j)$ is the number of children of super-node c_j. Let $B = \sum_v p(v)$ be an upper bound on the maximum achievable profit. Once the entire table $\chi[\mathrm{c}, \sigma, j, \beta]$ is computed for all $\mathrm{c} \in \mathrm{V}$ and $0 \leq \beta \leq B$, we can extract the answer from the table as follows. The maximum attainable profit is simply the largest value of β such some entry $\chi[\mathrm{root(T)}, \sigma, degree(\mathrm{root(T)}), \beta]$ is at most the server capacity P.

The run time of our dynamic programming solution can be evaluated as follows. Without loss of generality, there are $O(n)$ super-nodes and super-edges in the cluster-tree since each super-node can be assumed to have more than one child, where n is the number of VMs. The χ table has $O(2^k nB)$ entries, since each super-node contains at most k nodes, the number of super-edges is $O(n)$, and B is an upper bound on the maximum achievable profit. Each entry can be computed in $O(2^k B)$ time. Thus, the total running time of the algorithm is $O(2^{2k} nB^2)$, which is $O(nB^2)$ if we assume that the width of the cluster-tree k is a constant.

THEOREM 3.2.1. *For the single-server VM maximization problem in the cluster-tree sharing model, our algorithm produces a solution that is at least $(1-\epsilon)$ of optimal with a run time of $O(n^5/\epsilon^2)$, where n is the number of VMs. That is, there exists an FPTAS for the VM maximization problem in the cluster-tree sharing model.*

PROOF. We can extend the dynamic programming solution to create an FPTAS for single-server VM maximization. Analogous to the FPTAS for the knapsack problem [7], we derive an approximation by rounding the profit values of the VMs. For each node v, we round the profit values to create a new profit value $\tilde{p}(v) = \left\lfloor \frac{p(v)}{K} \right\rfloor$ for some $K = \epsilon \frac{p_{max}}{n}$, where $p_{max} = \max_v p(v)$. With the rounded profit values, the maximum profit $\tilde{B} = O(\frac{1}{\epsilon} n^2)$. The running time is then $O(n\tilde{B}^2) = O(n\left(\frac{1}{\epsilon} n^2\right)^2) = O(n^5/\epsilon^2)$.

Let $\tilde{X}$ be the solution obtained by our approximation algorithm using the rounded profit values. The profit obtained by solution $\tilde{X}$ is $\tilde{z} = \sum_{v \in \tilde{X}} p(v)$. Likewise, let $X^\star$ be the optimal solution obtaining a profit of $z^\star = \sum_{v \in X^\star} p(v)$.

$$\tilde{z} = \sum_{v \in \tilde{X}} p(v) \geq \sum_{v \in \tilde{X}} K \left\lfloor \frac{p(v)}{K} \right\rfloor \geq \sum_{v \in X^\star} K \left\lfloor \frac{p(v)}{K} \right\rfloor \quad (1)$$

$$\geq \sum_{v \in X^\star} K \left(\frac{p(v)}{K} - 1 \right) = \sum_{v \in X^\star} (p(v) - K) = z^\star - |X^\star| K, \quad (2)$$

where the last inequality in Equation 1 follows from the fact that $\tilde{X}$ is the optimal solution for the rounded profit values and hence obtains at least as much profit as $X^\star$. It follows from Equation 2 and the fact that $K = \epsilon p_{max}/n \leq \epsilon z^\star / |X^\star|$,

$$\frac{z^\star - \tilde{z}}{z^\star} \leq \frac{|X^\star| K}{z^\star} \leq \epsilon.$$

Thus, we have proven the theorem. □

We now generalize the result for the single-server VM maximization problem to the VM maximization problem where we have $m \geq 1$ physical servers for hosting the VMs. We utilize the results in [4] for the Separable Assignment Problem (SAP) where the authors show that a β-approximation for a single-server problem can be converted using LP-rounding to a $\left(1 - \frac{1}{e}\right)$ β-approximation algorithm for the multi-server problem. Further, in the cases where the single server problem admits an FPTAS, the result can be strengthened to provide an approximation ratio of $(1 - \frac{1}{e})$. Thus, we can state the following theorem.

THEOREM 3.2.2. *For the multi-server VM maximization problem in the cluster-tree sharing model, there exists a poly-time algorithm that produces a solution that is at least $(1-\frac{1}{e})$ of optimal, where n is the number of VMs and e is the transcendental number.*

3.2.2 VM packing

In the VM packing problem, we are given a set of virtual machines $V = \{v_1, v_2, \cdots v_n\}$, where each virtual machine v_i contains a set of pages P_i. The goal is to allocate all VMs to servers such that the memory capacity P of each server is not exceeded and the total number of servers are minimized.

First, we consider the VM packing problem in the cluster-tree sharing model. We can apply our algorithm for VM Maximization to derive a solution for VM packing as follows.

1. Set the profit value of all VMs to be 1. Run the VM maximization algorithm with the number of servers

m successively set to $1, 2, 2^2, \cdots 2^i, \cdots$, until at least $(1-\frac{1}{e})n$ VMs are successfully packed by the algorithm.

2. Let m^* be the number of servers where step (1) succeeds. Repeatedly run our VM Maximization algorithm on the remainder of the unpacked VMs using m^* servers each time, until no more VMs are left.

THEOREM 3.2.3. *The above algorithm runs in polynomial time and achieves a solution that is within $O(\log n)$ factor of optimal for the VM packing problem on a cluster-tree* T *with n VMs.*

PROOF. Let OPT be the minimum number of servers needed to pack cluster-tree T. For any $m \geq OPT$, we know that our VM maximization algorithm packs at least $(1-\frac{1}{e})n$ VMs into m servers, since each VM is assigned unit profit, the optimal profit is n, and Theorem 3.2.2 guarantees that our VM maximization algorithm achieves at least $(1-\frac{1}{e})$ of the optimal profit. Thus, the m^* achieved in the step (1) of the above algorithm is at most $2 \cdot OPT - 2$. Each time the VM maximization algorithm is run in step (2), the number of VMs left unpacked reduces by a factor of $\frac{1}{e}$. Thus, in at most $\lceil \ln n \rceil$ rounds all VMs will be packed. The total number of servers used is at most

$$m^* \lceil \ln n \rceil \leq (2 \cdot OPT - 2) \lceil \ln n \rceil = O(\log n) \cdot OPT.$$

Our algorithm for VM packing runs in polynomial time since our polynomial time algorithm for VM Maximization is invoked $O(\log n)$ times. □

For the simpler tree model, we now show that we can obtain a better approximation algorithm. Specifically, we present a 3-approximation algorithm for the VM packing problem in the tree model. We first compute a lower bound on the optimal number of servers needed to pack tree T. Next, we design a "greedy" algorithm that produces a packing solution for T that is within a factor 3 of this lower bound, and hence within a factor 3 of the optimal.

A fractional packing lower bound for the tree model.

One may think of the lower bound as the number of servers utilized by a "lower-bounding process" (LB process) that can fractionally pack VMs into servers. Since the LB process can split the pages associated with the nodes in tree T between multiple servers in a manner that a VM packing algorithm cannot, this fractional packing provides a lower bound on the number of servers that may or may not be achievable by a VM packing algorithm. Observe that the available server capacity varies as we move up the tree. The server capacity available at the root, denoted by $cap(root(\mathrm{T}))$, is the full server capacity of P. The capacity of any node v, $cap(v)$, can be inductively defined to be the residual capacity after packing all the ancestors of v that are required to be present at the server, i.e., $cap(v) = cap(parent(v)) - w(v)$. Alternately,

$$cap(v) = P - \sum_{v' \ is \ ancestor \ of \ v} w(v').$$

The LB process packs tree T in a bottom-up fashion as follows.

1. Pack each leaf v all by itself in a server of capacity $cap(v)$.

2. Suppose that a non-leaf node v has children v_i, $1 \leq i \leq l$. Inductively, suppose that each tree rooted at v_i, $1 \leq i \leq l$, has been packed "perfectly" by the LB process into servers of capacity $cap(v_i) = cap(v) - w(v)$ each. In a "perfect" packing all servers are full except possibly the last one. The LB process packs the tree rooted at v simply by consolidating the (at most l) partially-filled servers of its children into some number of full servers and at most one partially-filled server. This consolidation is easily achievable since the LB process is allowed to split any node across multiple servers. The full servers of its children are not repacked in any way. Finally, the server size is increased to $cap(v) = cap(v_i) + w(v)$ and a copy of v is added to each server to fill this increased capacity.

Let $size(v)$ represent the total size (in pages) required by the LB process to pack the subtree rooted at v. Note that $size(v)$ incorporates the weight of each node multiplied by the number of copies of the node that were made. Further, let the $count(v)$ denote the number of servers with capacity $cap(v)$ needed by the LB process to pack the tree rooted at v. Equivalently, $count(v)$ can be thought of as the number of copies of node v made by the LB process. The following inductive relationships hold. For each leaf v, $size(v) = w(v)$ and $count(v) = 1$. For each non-leaf node v with children v_i, $1 \leq i \leq l$,

$$count(v) = \left\lceil \frac{\sum_{i=1}^{l} size(v_i)}{cap(v) - w(v)} \right\rceil \quad (3)$$

$$size(v) = \sum_{i=1}^{l} size(v_i) + count(v) \cdot w(v) \quad (4)$$

THEOREM 3.2.4. *Any VM packing algorithm for tree* T *must pack at least $size(root(\mathrm{T}))$ pages and utilize at least $count(root(\mathrm{T}))$ servers.*

PROOF. We prove the theorem inductively starting from the leaves of T. In the base case, the theorem is clearly true for the leaves. Let a non-leaf node v have children v_i, $1 \leq i \leq l$. Assume inductively that any algorithm requires at least $size(v_i)$ pages to pack the trees rooted at v_i, for $1 \leq i \leq l$. Using Equation 3, we infer that any algorithm requires at least $count(v)$ copies of node v. Thus, using Equation 4, any algorithm must have size at least $size(v)$, since an additional $count(v)$ copies of v with $w(v)$ pages per copy are required. □

A greedy packing algorithm.

Our algorithm GREEDY packs tree T as follows.

1. Run the LB process to compute $size(v)$ and $count(v)$ for each node in T.

2. If the $count(root(\mathrm{T})) = 1$, then the entire tree T is packed into one server by the LB process. Likewise, the algorithm can pack the entire tree T into one server.

3. Else, if $count(root(\mathrm{T})) \geq 1$, do the following.

   (a) Let k be the smallest count value of some node in T such that $k > 1$. Pick a node v with count k such that all its children have a count value of 1. View the set of VMs that are descendants of

a child of v as a single item, resulting in as many items as the number of children of v. Note that each item fits into a single server of capacity P. Pack all items into servers of capacity P using any good bin packing algorithm, such as First Fit[2]. Now, all the VMs that are descendants of v have been packed.

(b) Remove the subtree rooted at v from T to form a new tree T′. Recursively, run GREEDY on T′.

LEMMA 3.2.5. *In step 3(a) of the* GREEDY *algorithm, the number of servers used to pack the tree rooted at v is at most $2 \cdot count(v) - 1$ servers.*

PROOF. Note that the First-Fit algorithm ensures at most one server is filled to half or less of its capacity, since any two such servers would have been combined by First-Fit into one server reducing the server count. Thus, since the LB process packs the tree rooted at v in $count(v)$ servers, our algorithm needs at most $2 \cdot count(v) - 1$ servers to pack the same contents, since otherwise two or more servers would be at most half full. □

THEOREM 3.2.6. *Algorithm* GREEDY *packs tree* T *using a number of servers that is within a factor of* 3 *of the optimal solution.*

PROOF. We use induction on $count(\text{root(T)})$. The base case of $count(\text{root(T)}) = 1$ is trivially true since GREEDY packs T in one server. Inductively, if $count(\text{root(T)}) > 1$, the tree T′ constructed in step 3(b) has a smaller count value than T. Note that the LB process constructs at least $count(v) - 1$ full servers at v. These full servers constructed by the LB process at node v will remain in its final packing of T unchanged, and is guaranteed to disappear when the tree rooted at v is removed to form T′. Thus,

$$count(\text{T}') \leq count(\text{T}) - count(v) + 1. \quad (5)$$

Applying the inductive hypothesis to T′, the algorithm packs T′ using at most $3 \cdot count(\text{root(T}'))$ servers. Additionally, by Lemma 3.2.5, the tree rooted at v in step 3(a) of the algorithm can be packed in $2 \cdot count(v) - 1$ servers. Thus, using Equation 5, the total number of servers used by the algorithm on tree T is at most

$$\begin{aligned} & 3 \cdot count(\text{T}') + 2 \cdot count(v) - 1 \\ \leq\ & 3 \cdot (count(\text{T}) - count(v) + 1) + 2 \cdot count(v) - 1 \\ \leq\ & 3 \cdot count(\text{root(T)}), \text{ since } count(v) \geq 2. \end{aligned}$$

□

4. EXPERIMENTAL EVALUATION OF VM PACKING

We experimentally evaluated the performance of our algorithm GREEDY for VM packing in the tree model. To quantify the packing benefits due to real-world sharing that occurs from actual usage, we only used traces from our 31 volunteer machines (which represent "real" workloads) and did not consider the 20 laboratory VMs (which represent synthetic workloads). Since we needed a large number of traces for the VM packing experiment, we extracted 4 traces from each of the 31 volunteer machines. The traces were spaced far enough apart in time so as to not be closely correlated. We used the 124 VM traces obtained in this manner in our VM packing experiments.

[2]While First Fit does not specify an order for processing the VMs, we have empirically observed that starting by placing two VMs that share the most pages in a server and consecutively picking the VM that shares the most with the most recent partially-packed server worked well in our implementation in Section 4.

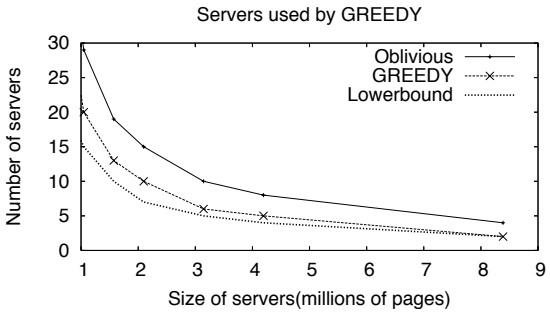


Figure 4: Comparison of server usage

We implemented GREEDY with a small twist. In step 3(a) of GREEDY when the First-Fit algorithm is applied on a set of VMs, we let our implementation of First-Fit utilize *all* available sharing between the VMs that are packed into a given server. That is, when First-Fit considers adding a new VM to a server, it assumes that any page present in the new VM that is already present in some VM in the server can be shared. Note that some of the pages shared in this fashion may not be explicitly captured in the tree model; We use this GREEDY variant since it better captures how page sharing works in actual virtualization platforms—while the sharing in the tree model guides the GREEDY VM packing, once VMs are co-located, the virtualization platform (i.e., hypervisor) will exploit *all* of the shared pages, and not just the subset identified by the tree model at packing time. Thus, this change lets our implementation capture the actual sharing benefit and not just the fraction identified by the tree model.

4.1 Server usage

The goal of VM packing is to minimize the number of servers needed to pack a given set of VMs. We compare the performance of GREEDY to that of a good sharing-oblivious algorithm. We implemented the Modified First Fit Decreasing (MFFD) algorithm [17] that is sharing-oblivious and is one of the best efficiently computable approximation schemes for bin packing. In addition, we compare GREEDY with a lower bound on the optimal number of servers required for packing the VMs. Note that the fractional packing lower bound derived in Section 3.2.2 applies to algorithms that only use the inter-VM sharing captured in the tree model. Since our implementation of GREEDY could use some additional sharing that is not captured in the tree model, it would be fairer to compare GREEDY with a lower bound derived for the general sharing model.

4.1.1 A server lower bound for general sharing

We derive two lower bounds and take the maximum of both bounds. Since these are lower bounds for general sharing, we make no assumptions about how the pages are shared.

The first lower bound is a simple size bound. Let *OPT*

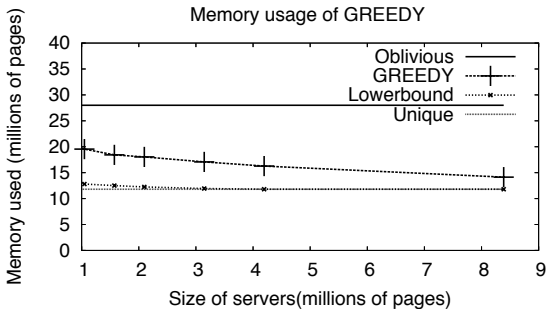


Figure 5: Comparison of memory usage

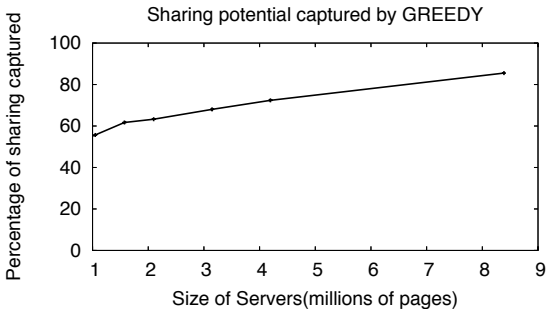


Figure 6: Realized sharing potential of GREEDY

be the smallest number of servers required to pack the given set of VMs, $V = \{v_1, v_2, \cdots v_n\}$. Let $UNIQ(V)$ be the set of unique pages contained in all the VMs in V, and let P be the server capacity. Clearly, $\lceil |UNIQ(V)|/P \rceil$ servers are needed to pack all the VMs.

Tp derive the second lower bound, we create a new set of VMs $V' = \{v'_1, v'_2, \cdots v'_n\}$ from the original set of VMs $V = \{v_1, v_2, \cdots v_n\}$ by removing each shared page from all but one of the VMs that contain it. Note that the optimal number of servers required to pack V' (call it OPT') is a lower bound on OPT. Further, since the VMs in V' share no pages, they can be packed with a good bin packing algorithm such as MFFD. Let $MFFD(V')$ denote the number of servers used by MFFD for the VMs in V'. From [17], we know that $MFFD(V') \leq \frac{71}{60} OPT' + 1$. Thus,

$$OPT \geq OPT' \geq \left\lceil \frac{60}{71}(MFFD(V') - 1) \right\rceil$$

.

Combining the two bounds, we have

$$OPT \geq \max \left\{ \lceil |UNIQ(V)|/P \rceil, \left\lceil \frac{60}{71}(MFFD(V') - 1) \right\rceil \right\}.$$

4.1.2 Results

Figure 4 shows the performance of GREEDY in comparison with the performance of sharing-oblivious MFFD (marked "Oblivious" in the figure) and the lower bound derived in Section 4.1.1. As the server memory capacity increases, we see a decrease in the number of servers and a decrease in the gap between the performance of GREEDY and the lower bound. It also shows that GREEDY is within 20% to 43% of the lower bound and 32% to 50% more efficient than the sharing-oblivious MFFD scheme. Furthermore, regardless of server size, GREEDY significantly reduces the number of servers required compared to a sharing oblivious algorithm such as MFFD. We see that as the server size increases the relative performance of GREEDY to MFFD increases from 32% to 50%, since the larger server sizes allow more sharing to be exploited by GREEDY.

4.2 Memory usage

We now show that GREEDY exploits inter-VM page sharing to decrease the total memory footprint.

4.2.1 A memory lower bound for general sharing

We derive a lower bound on the total memory footprint required for packing a set of VMs $V = \{v_1, v_2, \cdots v_n\}$ in the general sharing model. For any $V' \subseteq V$, let $UNIQ(V')$ denote the set of unique pages contained in the VMs in set V'. For any page p, let $V_p = \{v \in V : VM\ v\ contains\ page\ p\}$. Note that each page p must have at least $\lceil |UNIQ(V_p)|/P \rceil$ replicas in any VM packing of V, where P is capacity of the server. Thus, the following expression is a lower bound on the total memory footprint of any packing of V:

$$\sum_{p \in UNIQ(V)} \lceil |UNIQ(V_p)|/P \rceil .$$

4.2.2 Results

In Figure 5, we compare the memory usage of GREEDY with that of any sharing-oblivious algorithm. Note that any sharing-oblivious algorithm uses a memory footprint that equals the sum total of the sizes of all the VMs (marked "Oblivious" in the figure). In addition, we plot the lower bound for the memory footprint of any VM packing algorithm in the general sharing model that we derived in Section 4.2.1 (marked "Lowerbound" in the figure). We also plot $|UNIQ(V)|$, the number of unique pages in the set of VMs, which is a lower bound on the memory footprint of any algorithm independent of server memory capacity (marked "Unique" in the figure).

From Figure 5, we see that as the server memory capacity increases, the gap between the performance of GREEDY and the lower bound decreases. Further, GREEDY approaches the unique pages bound (i.e., perfect sharing) as the server memory capacity increases. Furthermore, we can see that GREEDY uses substantially less memory than any sharing-oblivious algorithm, specifically it uses between 25% to 57% fewer pages.

Another metric for analyzing the performance of GREEDY is by evaluating its realized sharing potential. Sharing potential is defined as the *maximum* achievable reduction in the memory footprint for a given server memory capacity P. Note that sharing potential is a non-decreasing function of P. As P tends to infinity, sharing potential tends to the difference between the total number of pages and the total unique pages in the set of VMs. Sharing potential is hard to compute exactly, since the computing the optimal reduction is itself NP-Hard. However, we can upper bound the sharing potential by using the lower bound on the memory footprint in Section 4.2.1. The *realized* sharing potential of a VM packing algorithm is percentage of the sharing potential that is actually achieved by the algorithm. A lower bound on the realized sharing potential of a VM packing algorithm can be computed using an upper bound for the sharing po-

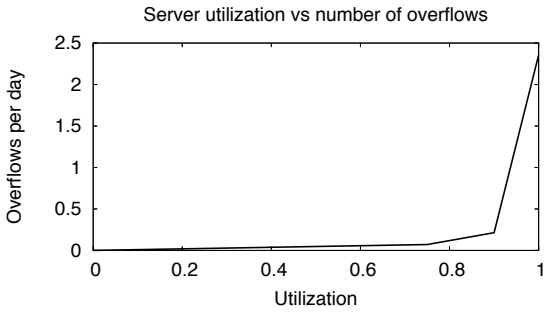


Figure 7: Memory overflows seen at different memory utilization levels.

tential. Using this process, we compute a lower bound on the realized sharing potential of GREEDY in Figure 6. We see that as the server memory capacity increases, GREEDY is able to realize an increasing percentage of the sharing potential, since it can pack more VMs on each server for larger server memory capacities, allowing for a larger amount of sharing to be captured. For around 4M memory pages, more than 70% of the sharing potential is realized by GREEDY.

4.3 Memory overflow

As sharing patterns and memory requirements of VMs change over time, the number of pages required on a server can become greater than its capacity, causing memory to become over-committed, a phenomenon that we term as "memory overflow". Memory overflow can cause excessive page faulting degrading the performance of the VMs. This might in turn trigger the potentially expensive operation of reallocating all the VMs in accordance with their current requirements and sharing characteristics. With this in mind, we study how stable the packings produced by GREEDY are over time. Our primary finding is that if we let GREEDY underutilize each server by a small percentage, server overflows become significantly rarer. The reason is that the extra unallocated memory capacity absorbs some of the variations caused by changing memory sharing characteristics.

We study the tradeoff between server memory utilization and overflow. For a given utilization u, $0 \leq u \leq 1$, algorithm GREEDY allocates VMs such that the memory capacity of each server is at most u times the actual server memory capacity. Using 3 physical servers with a capacity of 4194304 pages and the traces from the 31 volunteer machines, we found that almost all server overflows can be avoided by utilizing the servers to only 90% of capacity. Figure 7 shows us that for 90% utilization, we see only 3 faults over the course of a 2 week time frame. This gives an expected 4.67 days until a repacking of VMs due to overflow is required. On the other hand, fully utilizing the physical servers to 100% of its memory capacity results in 33 faults and an expected 0.42 days until reallocation is required.

5. RELATED WORK

Systems work. Content-based page sharing for virtual machines was implemented in the Disco system [1], incorporated in VMware ESX [14] and later in Xen [9]. In these systems, the hypervisor uses hashing and page comparison to identify and share identical pages running on the same physical server. There has been recent systems work in exploiting sharing at the sub-page level by sharing portions of a page such as the Difference Engine system [5]. While we explicitly consider only page-level sharing in our work, our models and results can be extended to account for sub-page level sharing as well.

While much of the work content-based page sharing focus on effectively sharing the memory contents of VMs located on the same physical server, the potential for exploiting inter-VM sharing by intelligently colocating VMs was first studied in [16]. In [16], the authors provide a compact fingerprinting scheme using bloom filters that can identify VMs with a large sharing potential, and show that "sharing aware" placement has the potential to significantly improve memory usage. Our work takes the next step by developing formal models and provably-good algorithms for sharing-aware placement of VMs on physical servers with the goal of achieving the potential benefits of inter-VM sharing.

In our work, we have focused on optimizing memory resources, whereas the general VM colocation also incorporates other server resources such CPU, disk, and network. For instance, the VMware Distributed Resource Scheduler [13] monitors CPU, network, and memory utilization in clusters of virtual machines and use migration for load balancing. We view our work as integrating into the larger multi-resource VM colocation framework by helping incorporate sharing-aware VM placement into these systems.

Algorithmic work. There has been no prior algorithmic work in the VM maximization problem per se, though we utilize algorithmic techniques similar to that used in the closely-related knapsack problem [11]. Note that the special case of VM maximization where VMs have no shared pages is the knapsack problem. Numerous variants of the knapsack problem have been studied over the years, particularly relevant is the partially-ordered knapsack problem where the items that are packed must obey precedence constraints expressed as a partial order [7, 10]. In fact, our FPTAS for the single-server VM maximization problem for cluster-tree sharing uses the "left-right" dynamic programming technique that was developed for the tree knapsack problem in [7].

A generalization of the knapsack problem where a submodular function describes the cumulative size of any collection of items was recently studied by Fleischer and Svitkina[12]. They present a bi-criteria randomized $\left(\sqrt{\frac{n}{\log n}}, \frac{1}{2}\right)$-approximation algorithm, where a (ρ, σ)-approximation is defined as a solution where the knapsack is allowed to overflow by a factor of ρ and the profit function is guaranteed to be within a factor of σ of optimal. Furthermore, they proved a lower bound on all approximation schemes for this problem of $\frac{\rho}{\sigma} = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$. While page sharing in VMs is submodular, it is much more approximable than the arbitrary submodular functions considered in [12].

A special case of VM packing where the VMs do not share any pages is the classical bin packing problem [2]. While dozens of variants of bin packing have been studied in the literature over the past decade, there is no prior algorithmic work to our knowledge that considers the interesting case where the cumulative size of the packed items can be smaller than the sum of the individual sizes due to sharing.

6. CONCLUSIONS AND OPEN PROBLEMS

In this paper, we initiated the study of sharing models and sharing-aware algorithms for VM colocation. Our work exposes the tradeoff between complex sharing models that capture all of the inter-VM sharing but are hard to exploit algorithmically, and structured hierarchical models that ignore some of the inter-VM sharing but are more amenable to provably-efficient algorithm design. Using actual VM traces, we demonstrated that hierarchical sharing models can capture a large percentage of the inter-VM sharing, making them a viable option for real-world VM colocation. Our experiments show that our VM packing algorithm exploits inter-VM sharing to substantially reduce number of servers and memory footprint and that these packings can be relatively stable over time.

Our work opens a number of exciting directions for future research. A key direction is tightening the approximation bounds, both better algorithms and lower bounds, for the VM packing problem (see Figure 1). For instance, is there an asymptotic PTAS for VM packing in the hierarchical sharing models, or even better approximation ratios? Bin packing, a special case of VM packing, has an asymptotic PTAS [3, 8], though variants are known not to have an asymptotic PTAS'es [15]. While our algorithms work in "batch mode", an important research direction is extending our work to an online setting where VM packing occurs as and when VMs are created and destroyed. Further, our work focused primarily on the server memory resource. Extending our work to multi-resource VM colocation that takes into account memory, CPU, and network resources is another important direction for future work. Finally, studying how our techniques can be applied to cloud computing platforms of the future is an interesting avenue for research. As modern data centers and cloud platforms evolve, so will the structural properties of VM sharing. Investigating these platforms may lead to new models and algorithms for reducing the operational cost of the virtualization service providers.

7. ACKNOWLEDGMENTS

We thank our shepherd Anne Benoit and the anonymous reviewers for their comments. We thank Sean Barker for his help with gathering synthetic laboratory traces and the numerous volunteers in our department for contributing memory traces for this study. The work of Michael Sindelar and Ramesh Sitaraman was supported in part by an NSF Award CNS-05-19894. Much of the work of Michael Sindelar was done while he was at UMass, Amherst. The work of Prashant Shenoy was supported in part by NSF grants OCI-1032765, CNS-0916972 and CNS-0720616.

8. REFERENCES

[1] E. Bugnion, S. Devine, and M. Rosenblum. DISCO: Running Commodity Operating Systems on Scalable Multiprocessors. In *SOSP*, pages 143–156, 1997.

[2] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-hard Problems*, pages 46–93, Boston, MA, USA, 1997. PWS Publishing Co.

[3] W. Fernandez de la Vega and G. Lueker. Bin packing can be solved within 1+ ε in linear time. *Combinatorica*, 1(4):349–355, 1981.

[4] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proceedings of the Seventeenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 611–620, New York, NY, USA, 2006. ACM.

[5] 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. In *Usenix OSDI*, December 2008.

[6] M. Hajiaghayi, K. Jain, K. Konwar, L. Lau, I. Mandoiu, A. Russell, A. Shvartsman, and V. Vazirani. The minimum k-colored subgraph problem in haplotyping and DNA primer selection. In *Proceedings of the International Workshop on Bioinformatics Research and Applications (IWBRA)*, 2006.

[7] D. Johnson and K. Niemi. On knapsacks, partitions, and a new dynamic programming technique for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.

[8] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science*, pages 312–320. IEEE, 1982.

[9] J. Kloster, J. Kristensen, and A. Mejlholm. On the Feasibility of Memory Sharing: Content-Based Page Sharing in the Xen Virtual Machine Monitor. Master's thesis, Department of Computer Science, Aalborg University, June 2006.

[10] S. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155(8):889–897, 2007.

[11] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. Wiley & Sons, 1990.

[12] Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, pages 697–706. IEEE Computer Society, 2008.

[13] VMware. DRS performance and best practices. 2008.

[14] C. Waldspurger. Memory Resource Management in VMWare ESX Server. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, Dec. 2002.

[15] G. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293–297, 1997.

[16] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M. Corner. Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers. In *2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009)*, Washington, DC, USA, March 2009.

[17] M. Yue and L. Zhang. A simple proof of the inequality MFFD(L) $\leq$ 71/60 OPT(L) + 1,L for MFFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica (English Series)*, 11:318–330, July 1995.