Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers

Timothy Wood, Gabriel Tarasuk-Levin, Prashant Shenoy,
Peter Desnoyers†, Emmanuel Cecchet, Mark D. Corner

Department of Computer Science, Univ. of Massachusetts Amherst
† Department of Computer & Information Science, Northeastern University
[twood,gtarasuk,shenoy,cecchet,mcorner]@cs.umass.edu, pjd@ccs.neu.edu

Abstract

Many data center virtualization solutions, such as VMware ESX, employ content-based page sharing to consolidate the resources of multiple servers. Page sharing identifies virtual machine memory pages with identical content and consolidates them into a single shared page. This technique, implemented at the host level, applies only between VMs placed on a given physical host. In a multi-server data center, opportunities for sharing may be lost because the VMs holding identical pages are resident on different hosts. In order to obtain the full benefit of content-based page sharing it is necessary to place virtual machines such that VMs with similar memory content are located on the same hosts.

In this paper we present Memory Buddies, a memory sharing-aware placement system for virtual machines. This system includes a memory fingerprinting system to efficiently determine the sharing potential among a set of VMs, and compute more efficient placements. In addition it makes use of live migration to optimize VM placement as workloads change. We have implemented a prototype Memory Buddies system with VMware ESX Server and present experimental results on our testbed, as well as an analysis of an extensive memory trace study. Evaluation of our prototype using a mix of enterprise and e-commerce applications demonstrates an increase of data center capacity (i.e. number of VMs supported) of 17%, while imposing low overhead and scaling to as many as a thousand servers.

Categories and Subject Descriptors D4-7 [*Organization and Design*]: Distributed systems; D4-2 [*Storage Management*]: Main Memory; D4-8 [*Performance*]: Measurements

General Terms Design, Management, Measurement

Keywords Virtualization, Page sharing, Consolidation

1. Introduction

Data centers—server farms that run networked applications—have become popular in a variety of domains such as web hosting, enterprise applications, and e-commerce sites. Modern data centers are increasingly employing a virtualized architecture where applications run inside virtual servers mapped onto each physical server in the data center. These virtual machines (VMs) run on top of a *hypervisor*, which is responsible for allocating physical resources such as memory and CPU to individual VMs.

To intelligently share RAM across VMs, modern hypervisors use a technique called *content-based page sharing (CBPS)* [27; 11]. In this technique, duplicate copies of a page resident on a host are detected and a single copy of the page is shared, thereby reducing the memory footprint of resident VMs.

The concept of transparent page sharing was first proposed in the Disco system [3] as a way to eliminate redundant copies of pages (typically code pages) across virtual machines. While Disco's transparent page sharing required modifications to the guest OS, VMware ESX Server introduced content-based page sharing [27], which identifies and shares duplicate pages by examining their contents transparent to the guest operating system. Today this technique is widely deployed in VMware ESX, with experimental support in Xen. The potential benefits of content-based page sharing are well documented; for instance, the original VMware ESX paper [27] reports memory savings of as much as 33% in measured production environments. Support for memory sharing at finer, sub-page granularity can save more than 65% [8].

However, a CBPS sharing mechanism by itself only shares redundant pages *after* a set of VMs have been placed onto a physical host—the mechanism does not address the problem of *which* VMs within the data center to colocate onto each host so that page sharing can be maximized. Thus, to fully realize the benefits of this mechanism, a data center should implement an intelligent colocation strategy that identifies virtual machines with high sharing potential and then maps them onto the same host. Such a colocation strategy can be employed both during the initial placement of a new VM as well as during a server consolidation phase in order to consolidate existing VMs onto a smaller number of physical hosts.

In this paper we present *Memory Buddies*, a system for intelligent VM colocation within a data center to aggressively exploit page sharing benefits. The key contribution of this work is a memory fingerprinting technique that allows our system to identify VMs with high page sharing potential. The memory fingerprints produced are compact representations of the memory contents of virtual machines; these fingerprints may be compared to determine the number of redundant pages between VMs and thus the potential for memory savings. We present two memory fingerprinting techniques: an exact hash list-based approach and a more compact Bloom filter-based approach, representing a trade-off between computational overhead and accuracy/space efficiency.

Our second contribution is an intelligent VM colocation algorithm that utilizes our memory fingerprinting techniques to identify VMs with high page sharing potential and colocate them onto the same host. We show how our colocation algorithm can be employed for initial placement of new VMs onto a data center's servers. We

VEE'09, March 11–13, 2009, Washington, DC, USA.

also demonstrate how our method can be employed to consolidate the existing VMs in a data center onto a smaller number of hosts using live VM migration. In addition to initial placement, our system also implements a memory "hotspot" mitigation algorithm to adapt to changes in application mix or behavior which may necessitate revisiting earlier placement decisions.

These mechanisms can be employed for colocating both virtualized server applications and virtualized desktops for thin-client computing applications. In addition to the online uses described above, our colocation strategies can also be employed for offline planning as well. For instance, in cases such as deployment of desktop virtualization, where existing non-virtualized applications will be migrated to a virtualized environment, our smart colocation algorithms provide an accurate estimate of the physical servers and memory required.

Our third contribution is the use of real memory usage data from nearly two dozen Linux and Mac OS X servers and desktops in our department. In addition to implementing our techniques using VMware ESX Server on a prototype Linux data center, we have deployed the fingerprinting portion of our implementation on these systems. Memory trace data collected from these real-world systems drive many of our experimental evaluations. These evaluations validate our approach of using memory fingerprints to drive smart colocation of virtual machines, with minimal run-time overhead. In our experiments we observe increases in data center capacity of 17% over naive placement, as measured by the number of VMs supported, merely by placing VMs in such a way as to maximize the potential of existing page sharing mechanisms.

2. Background and System Overview

Consider a typical virtualized data center where each physical server runs a hypervisor and one or more virtual machines. Each VM runs an application or an application component and is allocated a certain slice of the server's physical resources such as RAM and CPU. All storage resides on a network file system or a storage area network, which eliminates the need to move disk state if the VM is migrated to another physical server [5].

The hypervisor uses a content-based page sharing mechanism, which detects duplicate memory pages in resident VMs and uses a single physical page that is shared by all such VMs. If a shared page is subsequently modified by one of the VMs, it is unshared using copy-on-write [27]. Thus, if VM_1 contains M_1 unique pages, and VM_2 contains M_2 unique pages, and S of these pages are common across the two VMs, then page sharing can reduce the total memory footprint of two VMs to $M_1 + M_2 - S$ from $M_1 + M_2$. The freed up memory can be used to house other VMs, and enables a larger set of VMs to be placed on a given cluster.

Problem formulation: Assuming the above scenario, VM colocation problem is one where each VM is colocated with a set of other "similar" VMs with the most redundant pages. Several instantiations of the smart colocation problem arise during: (i) *initial placement*, (ii) *server consolidation* and (iii) *offline planning*.

During initial placement the data center servers must map a newly arriving VM onto existing servers so as to extract the maximum page sharing. For server consolidation, VMs need to be repacked onto a smaller number of servers (allowing the freed up servers to be retired or powered off). Offline planning is a generalization of initial placement where a set of virtual machines must be partitioned into subsets and mapped onto a set of physical server to minimize the total number of servers.

In each case, the problem can be reduced to two steps: (i) identify the page sharing potential of a VM with several candidate VM groups and (ii) pick the group/server that provides the best sharing/memory savings. In scenarios such as server consolidation, live migration techniques will be necessary to move each VM to its new home (server) without incurring application down-time [18; 5].

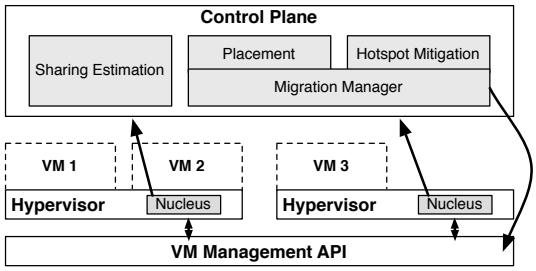


Figure 1. Memory Buddies System Architecture. The Nucleus sends memory fingerprint reports to the Control Plane, which computes VM placements and interacts with each hosts to place or migrate VMs according.

Finally, any data center that aggressively exploits page sharing should also implement hotspot mitigation to address any significant *loss* of page sharing due to application termination or major application phase changes—such loss of page sharing can create memory pressure and cause swapping. Hotspot mitigation techniques offload VMs to other servers to reduce memory pressure.

System Overview: Low-level page sharing mechanisms only detect and share duplicate pages belonging to resident VMs—they do not address the problem of *which* VMs to colocate on a host to maximize sharing. *Memory Buddies* detects sharing potential between virtual machines and then uses the low-level sharing mechanisms to realize these benefits.

Our system, which is depicted in Figure 1 consists of a *nucleus*, which runs on each server, and a *control plane*, which runs on a distinguished control server. Each nucleus generates a memory fingerprint of all memory pages within the VMs resident on that server. This fingerprint represents the page-level memory contents of a VM in a way which allows efficient calculation of the number of pages with identical content across two VMs. In addition to per-VM fingerprints, we also calculate aggregate per-server fingerprints which represent the union of the VM fingerprints of all VMs hosted on the server, allowing us to calculate the sharing potential of candidate VM migrations.

The control plane is responsible for virtual machine placement and hotspot mitigation. To place a virtual machine it compares the fingerprint of that VM against server fingerprints in order to determine a location for it which will maximize sharing opportunities. It then places the VM on this server, initiating migrations if necessary. The control plane interacts with VMs through a VM management API such as VMware's Virtual Infrastructure or the libvirt API [14].

The following sections describe the memory fingerprinting and control plane algorithms in detail.

3. Memory Fingerprinting

The nucleus runs on each physical server, computing memory fingerprints for each VM resident on that server, as well as for the server as a whole. Ideally the nucleus would be implemented at the hypervisor level, allowing re-use of many mechanisms already in place to implement content-based page sharing. Our experiments were performed with VMware ESX Server, however, and so lacking source code access[1] we have implemented the fingerprinting aspect of the nucleus within each VM, as a paired guest OS kernel module and user-space daemon.

[1] Our initial efforts had focused on the open-source Xen platform, where it was possible to make experimental modifications to the hypervisor. However, Xen's page sharing implementation is experimental and not compatible with its live migration mechanism; since our work requires both mechanisms, the work presented in this paper makes use of VMware ESX Server.

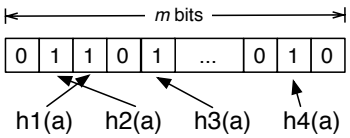


Figure 2. Bloom filter with four hash functions, containing a single key a.

3.1 Fingerprint Generation

Content-based page sharing implementations for both Xen and VMware ESX use hashes of page contents in order to locate pages with identical content which are thus candidates for sharing. In the Memory Buddies nucleus Hsieh's SuperFastHash algorithm [9] is used to generate 32 bit hashes for each 4KB page. In order to measure potential sharing *between* VMs, rather than *self-sharing*, or sharing within a single VM, we gather the set of unique page hashes for a VM's pages to generate the raw memory fingerprint. Maintained in sorted order, such a fingerprint may be compared against the fingerprint of another VM or server, yielding a count of the pages duplicated between the two VMs and thus the potential memory sharing between them.

3.2 Succinct Fingerprints

The memory fingerprints we have described consist of a list of page hashes; the intersection between two such fingerprints may be computed exactly, but they are unwieldy to use. Not only are they large—e.g. 1 MByte of fingerprint for each 1 GB of VM address space—but they must be sorted in order to be compared efficiently. To reduce this overhead, we also provide a *succinct fingerprint* which represents this set of hashes using a Bloom filter. [1; 13]. A Bloom filter is a lossy representation of a set of keys, which may be used to test a value for membership in that set with with configurable accurracy. The filter parameters may be set to trade off this accuracy against the amount of memory consumed by the Bloom filter.

As shown in Figure 2, a Bloom filter consists of an m-bit vector and a set of k hash functions $H = h_1, h_2, h_3, ..., h_k$ ($k = 4$ in the figure). For each element a, the bits corresponding to positions $H(a) = h_1(a), h_2(a), ..., h_k(a)$ are set to 1; to test for the presence of a, we check to see whether all bits in $H(a)$ are set to 1. If this test fails we can be certain that a is not in the set. However, we observe that the test may succeed—i.e. result in a *false positive*—if all bits in $H(a)$ were set by the hashes of some other combination of variables. The probability of such errors depends on the size of the vector m, the number k of bits set per key, and the probability that any bit in the vector is 1.

If the number of elements stored is n, the probability 'p_e' of an error when testing a single key against the filter is given by $p_e = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.
Thus given n, the number of pages belonging to a VM, proper choice of the size of the bit vector m and the number of hash functions k can yield a sufficiently small error probability. Two Bloom filters may be compared to estimate the size of the intersection of their key sets; this is covered in more detail in Section 3.3. In addition, multiple Bloom filters can be combined by taking the logical OR of their bit vectors; Memory Buddies uses this to create aggregate fingerprints for all VMs running on each host.

While Memory Buddies supports both fingerprint mechanisms, we note that in practice neither full hash lists nor succinct fingerprints will produce an absolutely accurate prediction of page sharing behavior, for several reasons. First, the fingerprints are snapshots of time-varying behavior, and lose accuracy as the actual memory contents change after the fingerprint was taken. A second reason is that comparison of snapshots only indicates how many pages *could* be shared; for various reasons the page sharing logic within the hypervisor may fail to share some pages which might otherwise be sharable.

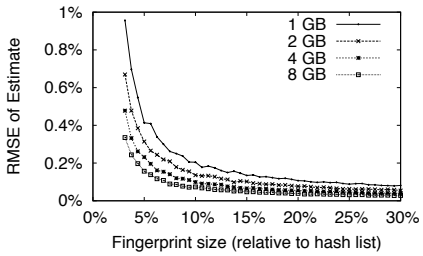


Figure 3. Root Mean Squared Error (RMSE) of page sharing estimate for different memory and Bloom filter sizes.

3.3 Fingerprint Comparison

To estimate page sharing potential, we need to compare the memory fingerprints of two or more virtual machines and compute their intersection: i.e. the number of identical pages between the two. With raw memory fingerprints consisting of the full list of memory page hashes, this may be done by sorting the lists, comparing them, and counting the number of matches. Comparing two concise fingerprints is somewhat more complicated, although faster.

To calculate the size of this intersection, we examine the case of two Bloom filters holding u_1 and u_2 unique entries each, plus c entries common between the two. We then take the bitwise AND of the two filter vectors; the elements of this resulting vector will be 1 for (a) each bit set by the c common elements, and (b) each bit set by one or more keys in each of the unique sets. We omit the mathematical derivation, which can be found in related work [2], but note that the expected number of shared elements is [15]:

$$share = \frac{ln(z_1 + z_2 - z_{12} - ln(z_1 * z_2) + ln(m)}{k(ln(m) - ln(m-1))} \quad (1)$$

where z_1 and z_2 are the numbers of zeros in the two Bloom filters, z_{12} is the number of zeros in the AND of the two filters, m is the size of each of the filter vectors, and k is the number of hash functions used.

The estimate contains a correction for the expected number of false matches between the two vectors, and is thus considerably more accurate than the test of a single key against the same Bloom filter. No closed-form solutions for the accuracy of this estimate have been derived to date; however we are able to measure it for particular cases via Monte Carlo simulation. In Figure 3 we see error results for different numbers of keys and Bloom filter sizes; the number of keys is expressed as a total amount of memory (i.e. 1 GB = 256K page hashes or keys) and the filter size is expressed as a fraction of the size needed for the full hash list (i.e. 256K hashes requires 1024KB at 4 bytes per hash). The error rate is the percent of total pages which are incorrectly considered to be shared or unshared by the Bloom filter. We see that with a filter as small as 5% of the size of the hash list—i.e. only slightly more than 1 bit per page—the expected error is less than 0.5%, allowing us to estimate sharing quite precisely with succinct fingerprints. In addition to the savings in communication bandwidth for reporting these succinct fingerprints, they are also much faster to compare, as they are both much smaller and, unlike hash lists, require no sorting before comparison.

4. Sharing-aware Colocation

The VM and server fingerprints are periodically computed and transmitted to the control plane by each nucleus; the control plane thus has a system-wide view of the fingerprints of all VMs and servers in the data center. The control plane implements a colocation algorithm that uses this system-wide knowledge to identify servers with the greatest page sharing potential for each VM that needs to be placed.

The control plane provides support for three types of placement decisions: initial placement of new VMs, consolidation strategies for live data centers, and offline planning tools for data center capacity planning.

4.1 Initial Placement

When a new virtual machine is added to a data center, an initial host must be selected. Picking a host based simply based on current resource utilization levels can be inefficient. The placement algorithm in Memory Buddies instead attempts to deploy VMs to the hosts which will allow for the greatest amount of sharing, reducing total memory consumption, allowing more VMs to be hosted on a given number of servers.

Each new VM is initially placed on a staging host where its resource usage and memory fingerprint can stabilize after startup and be observed. Each VM periodically reports memory fingerprints as well as the resource usages on each server. Monitored resources include memory, CPU, network bandwidth and disk; both the mean usage over the measurement interval as well as the peak observed usage are reported. The placement algorithm uses these reported usages to identify the best candidates for placing each new VM.

The algorithm first determines the set of feasible hosts in the data center. A feasible host is one that has sufficient available resources to house the new VM—recall that each VM is allocated a slice of the CPU, network bandwidth and memory on the host, and only hosts with at least this much spare capacity should be considered as possible targets. Given a set of feasible hosts, the algorithm must estimate the page sharing potential on each host using our fingerprint comparison technique—the fingerprint for the VM is compared with the composite fingerprint of the physical server directly using hash lists, or the number of shared pages is estimated using Equation 1 if compact Bloom filters are being used. The algorithm then simply chooses the feasible server that offers the maximum sharing potential as the new host for that VM.

4.2 Server Consolidation

Memory Buddies' server consolidation algorithm opportunistically identifies servers that are candidates for shutting down and attempts to migrate virtual machines to hosts with high sharing opportunities. In doing so, it attempts to pack VMs onto servers so as to reduce aggregate memory footprint and maximize the number of VMs that can be housed in the data center. Once the migrations are complete, the consolidation candidates can be retired from service or powered down until new server capacity is needed, thereby saving on operational (energy) costs. The consolidation algorithm comprises three phases:

Phase 1: *Identify servers to consolidate.* The consolidation algorithm runs periodically (e.g., once a day) and can also be invoked manually when needed. A list of hosts which are candidates for consolidation is determined by examining memory utilization statistics for each host; a server becomes a candidate for consolidation if its mean usage remains below a low threshold for an extended duration.[2] Currently our system only considers memory usages when identifying consolidation candidates; however, it is easy to extend it to check usages of all resources to identify lightly loaded servers.

[2] In addition, the system can also check that the peak usage over this duration stayed below a threshold, to ensure that the server did not experience any load spikes during this period.

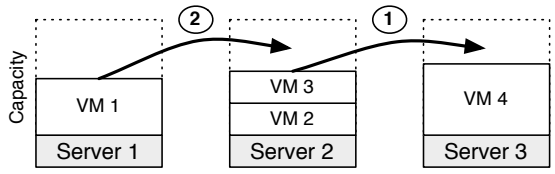


Figure 4. A 2-step migration: VM_3 is first migrated from Server 2 to Server 3 to free up space for VM_1. VM_1 can then be migrated from Server 1 to Server 2.

Phase 2: *Determine target hosts.* Once the set of consolidation candidates has been identified, the algorithm must determine a new physical server to house each VM. To do so, we order VMs in decreasing order of their memory sizes and consider them for migration one at a time. For each VM, the algorithm first determines the set of feasible servers in the data center as described in Section 4.1. The host which will provide the greatest level of sharing (while still ensuring sufficient resources) is then selected for each VM.

In certain cases, it is possible that there are no feasible servers for a VM. This can happen if the VM has a large CPU, network or memory footprint and existing servers in the data center are heavily utilized. In this case, the consolidation algorithm must consider a multi-way move, where one or more VMs from an existing server are moved to other servers to free up additional capacity and make this server feasible for the VM under consideration, as illustrated in Figure 4. As migration does impose some overhead, the algorithm attempts to minimize the number of moves considered in multi-way planning.

Phase 3: *Migrate VMs to targets.* Once new destinations have been determined for each VM on the consolidation servers, our algorithm can perform the actual migrations. Live migration is used to ensure transparency and near-zero down-times for the application executing inside the migrated VMs.

To ensure minimum impact of network copying triggered by each migration on application performance, our algorithm places a limit on the number of concurrent migrations; once each migration completes, a pending one is triggered until all VMs have migrated to their new hosts. The original servers are then powered off and retired or moved to a shutdown pool so they can be reinitialized later if memory requirements increase.

4.3 Offline Planning Tool for Smart VM Colocation

The Memory Buddies system can also be used for offline planning to estimate the required data center capacity to host a set of virtual machines. The planning tool can be used to answer "what if" questions about the amount of sharing potential for different VM configurations, or to generate initial VM placements.

The offline planner takes as input:

1. A list of the data center's hosts and their resource capacities.
2. Resource utilization statistics (CPU, network, and disk) for each system to be placed.
3. Memory fingerprints for each system.

The systems to be placed within the data center may either already be running as virtual machines, or may be sets of applications currently running on physical hosts which are to be moved to a virtualized setting (e.g. desktop virtualization). If the systems to be hosted in the data center are not yet running on virtual machines, then additional modeling techniques may be required to estimate the resource requirements of the applications after virtualization

overheads are added [30]. In either case, the memory fingerprints are gathered by deploying the memory tracer software (our kernel module implementation of the nucleus) on each system to be moved to the data center.

The planning tool can be used to analyze "what if" scenarios where a data center administrator wants to know about the resource consumption of different sets of VMs hosted together. The tool can output the amount of both inter-VM and self-sharing likely to occur for a given set of VMs. This provides valuable information about the expected amount of memory sharing from colocating different applications or operating systems.

The offline planner can generate VM placements that match each VM to a host such that the resource capacities of the host are not violated, while maximizing the amount of sharing between VMs. This is analogous to a bin packing problem where the resource constraints define the size of each bin. A variety of heuristics can be used for this sort of problem. Memory Buddies uses a dynamic programming technique which determines what subsets of VMs will fit on each host to maximize sharing while respecting constraints. These constraints may be simple resource consumption thresholds such as not using more than 80% of the CPU or requiring a portion of the server's memory to be kept free to prevent changes in sharing or memory requirements causing hotspots. Constraints can also be used to enforce business rules such as only colocating a single customer's VMs on a given host or to ensure fault tolerance by preventing multiple replicas of an application from being placed together. The tool's output provides a list of which VMs to place on what hosts, as well as the total memory consumption and expected rate of sharing.

5. Hotspot Mitigation

The Memory Buddies hotspot mitigation technique works in conjunction with the consolidation mechanism to provide a sharing-aware mechanism for resolving memory pressure caused by changes in virtual machine behavior. Our system must detect such hotspots when they form and mitigate their effects by re-balancing the load among the physical hosts. We note that memory hotspots are only one form of such overload; other kinds of hotspots can occur due to demand for CPU, network, and disk resources. Such overloads are best dealt with by other techniques [31] and are not considered in this work.

A memory hotspot may arise for several reasons. First, it may be due to increased demand for memory by one or more virtual machines. Changing behavior on the part of applications or the guest OS (e.g. the file system buffer cache) may result in a need for more memory, which the hypervisor will typically attempt to meet by contracting the memory balloon and returning memory to the guest. The second possible cause is due to a loss of page sharing. If changes in virtual machine behavior cause its memory contents to change (a so-called "phase change") in such a way as to reduce memory sharing, then overall memory usage on a physical server may increase even though the amount of memory seen by each guest OS remains constant.

The control plane relies on statistics reported by the Memory Buddies nucleus to detect memory hotspots. If implemented at the hypervisor level, the nucleus would have direct access to information on the availability of physical memory; in our prototype we must instead infer this information from guest behavior and externally reported hypervisor statistics. In particular, we monitor both the level of swap activity on each guest OS, as well as the number of shared pages reported by the hypervisor.

When swap activity rises above a certain threshold, a hotspot is flagged by the control plane, which then attempts to resolve it by re-distributing VMs among physical servers. In choosing a VM to move and a destination for that VM, we use the same algorithm as for initial placement. In particular, we examine all VMs on the overloaded system, and for each VM calculate the maximum gain in sharing which could be obtained by migrating that VM to another feasible host. We then choose to migrate the VM which provides the highest absolute gain in sharing—i.e. which provides the maximum system-wide increase in available memory.

If there are no feasible destinations for the virtual machines on the overloaded host, a server must be brought in from the shutdown pool so that it can host one or more of the VMs.

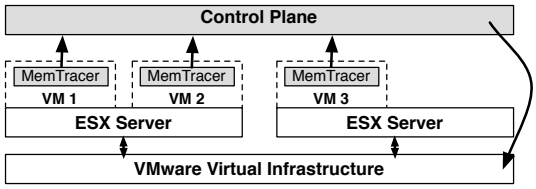


Figure 5. Our implementation uses VMware ESX since it supports migration and page sharing. As it is a closed source hypervisor, the nucleus is implemented as a memory tracer component running within each VM.

6. Implementation

The Memory Buddies implementation uses VMware ESX for the virtualization layer as it supports both page sharing and virtual machine migration. While the ESX hypervisor already gathers page hashes to determine sharable pages [27], this information is unavailable to our software because of ESX's closed nature. As a result, our system implementation moves the nucleus component from the hypervisor into each virtual machine in the form of a memory tracing kernel module that supports Linux, Windows, and Mac OS X. This software gathers the lists of hashes and sends them (or compact Bloom filters) to the control plane. Figure 5 illustrates the specifics of our implementation. We have deployed our tracer and control plane on a testbed for evaluation.

Memory Tracer: We have developed memory analysis tools that run on Linux, Mac OS X, and Windows. The Linux tracer supports both 2.4 and 2.6 kernels, the Mac tracer runs under OS X 10.5 on Intel and G4 systems, and the Windows tracer supports XP Service Pack 2 and 32-bit Vista systems. All of the tracers work by periodically stepping through the full memory of the machine being traced, generating 32 bit hashes for each page in memory. When used in our testbed, the tracer is run within each virtual machine, and the resulting hash lists (or Bloom filters) are sent to the control plane for processing every few minutes. An alternate version of our tracer has been designed solely for gathering memory traces, and has been distributed to volunteers within our department to be run on a variety of physical machines. We analyze the sharing potential in Section 7.2.

Under memory pressure, a VMware ESX host may make use of the *balloon driver* [27] to reclaim memory from running VMs. We note that as the guest OS in unaware of the balloon driver activity, our memory tracer may analyze such reclaimed memory pages. In practice, however, this is not a concern, as reclaimed memory will uniformly appear zeroed, and thus will not affect fingerprints based on unique page hashes.

Control Plane: The control plane is a Java based server which communicates with the VMware Virtual Infrastructure management console via a web services based API. The API is used by the control plane to discover which hosts are currently active and where each virtual machine resides. Extra resource statistics are retrieved from the VMware management node such as the total memory allocation for each VM. This API is also used to initiate virtual machine migrations between hosts. The control plane primarily consists of

statistic gathering, sharing estimation, and migration components which comprise about 3600 lines of code.

Memory Buddies Testbed: The testbed is a cluster of P4 2.4GHz servers connected over gigabit ethernet which combines the Control Plane with a set of virtual machines, each running the Memory Tracer. Each server runs VMware ESX 3.0.1 and the VMware Virtual Infrastructure 2.0.1 management system is on an additional node.

7. Experimental Evaluation

We have evaluated the Memory Buddies system to study the benefits of exploiting page sharing information when determining virtual machine placement.

Section 7.1 discusses our evaluation workloads and experiment specifications. Section 7.2 analyzes the sharing characteristics of the memory traces we have gathered. Our first case study measures the benefits of Memory Buddies for Internet Data Centers (section 7.3) on both our testbed and through trace driven simulation. Section 7.4 evaluates the hotspot mitigation algorithm on our testbed and we explore offline planning with a desktop virtualization case study in section 7.5. Finally, Section 7.6 shows the performance tradeoffs of the fingerprinting techniques available in Memory Buddies.

7.1 Experimental Workloads

We conduct two sets of experiments to evaluate Memory Buddies. First, we demonstrate the performance of our consolidation and hotspot mitigation algorithms on a small prototype Memory Buddies data center running realistic applications. Second, to demonstrate that these results apply to larger data centers and to real-world applications, we gather memory traces from live machines in our department and use these traces to evaluate the efficacy of our techniques.

Our prototype data center experiments are based on the following applications:

- RUBiS [4] is an open source multi-tier web application that implements an eBay-like auction web site and includes a workload generator that emulates users browsing and bidding on items. We use the Apache/PHP implementation of RUBiS version 1.4.3 with a MySQL database.
- TPC-W [23] models an Amazon style e-commerce website implemented with Java servlets and run on the Jigsaw server with a DB2 backend.
- SpecJBB 2005 [24] is a Java based business application benchmark which emulates a 3-tier system with a workload generator.
- Apache Open For Business (OFBiz) [20] is an open source suite of enterprise web applications with accouting, finance, and sales functionality used by many businesses. We utilize the eCommerce component and a workload generator based on the JWebUnit testing framework to emulate client browsing activities.

To ensure that inter-VM sharing is predominantly due to code pages, we randomize the data used by different instances of same application— the workloads and database contents are different for each VM instance to avoid sharing of data pages. For the multi-tier applications, we run all tiers within a single virtual machine. All Apache web servers are version 2.2.3 with PHP 4.4.4-9, MySQL databases are 5.0.41, Jigsaw is version 2.2.6, and the DB2 server was DB2 Express-C 9.1.2.

We extend our evaluation with a study of memory traces collected by our tracer tool. We have distributed the memory tracer application to volunteers within our department and gathered a total of over 130,000 memory fingerprints from more than 30 systems. [3] We use these traces both to analyze the sharing potential between actual systems and to allow emulation of larger data centers. Finally, we use these memory traces to analyze the accuracy and overhead of our different fingerprint comparison techniques.

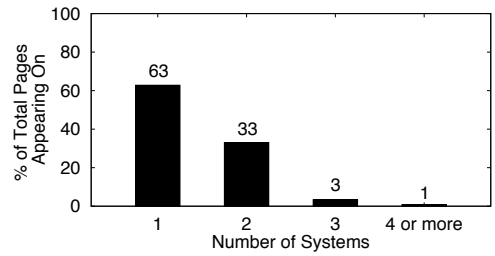


Figure 6. Percentage of memory pages duplication between VMs on a collection of 30 diverse laptops, desktops, and servers. 33% of pages were sharable with exactly one other machine, and 37% with one or more machines.

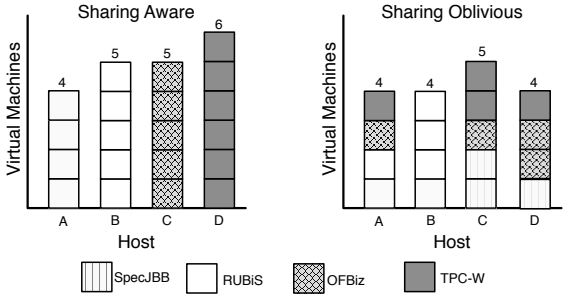


Figure 7. Sharing aware vs sharing oblivious placement. Sharing aware detects similar virtual machines and groups them on the same hosts.

7.2 Memory Trace Analysis

We have analyzed a subset of the traces gathered from machines within our department to provide a summary of the level of page sharing available across a diverse set of machines. Here we examine the fingerprints gathered on June 10th 2008 from 24 Linux and 6 Mac OS X systems (our collection of Windows traces is currently too small to provide significant results).

Figure 6 shows the number of pages which appear on only one, two, or more systems. This indicates that, as expected, the majority of pages are unique, only appearing on a single host. However, a significant portion of the pages reside on two or more machines. This suggests that in an ideal case where all systems could be colocated onto a single host, the total memory requirements of running these machines could be reduced by about 37%, giving an upper bound on the amount of sharing which could ever occur. We also note that while many pages appear on two systems (33%), very few reside on three or more machines. This emphasizes that random placement of virtual machines is unlikely to realize the full benefits of memory sharing.

7.3 Case Study: Internet Data Center

Many web hosting services rent virtual machine servers to customers since they can be much cheaper than providing dedicated servers. These virtual servers are an excellent candidate for exploit-

[3] We plan to release a portion of these traces and the code to process them to researchers worldwide on the UMass Trace Repository website.

Application	Measured Sharing	Pred. Sharing
TPC-W	38%	41%
OpenForBiz	18%	22%
RUBiS	16%	15%
SpecJBB	5%	5%

Table 1. Application types and their memory sharing levels. Measured sharing is obtained from the live statistics of the hypervisor, while predicted sharing is computed from the memory traces.

ing page sharing since the base servers often run similar operating systems and software, such as a LAMP stack or a J2EE environment. In this case study we first test Memory Buddies' ability to more effectively place different classes of applications typically found in an Internet data center. We utilize four different applications to vary the sharing rate between virtual machines, and a testbed with four hosts. Note that while the core application data is identical within an application class, the workloads and database contents are different for each VM instance. Table 1 lists the different application types and the level of sharing between pairs of virtual machines of the same type; actual sharing values vary within a few percent depending on paging activities. We present both the predicted sharing reported by our memory tracer and the actual level of sharing achieved by the hypervisor. For the first two applications, the predicted level of sharing is too high; this error occurs when the hypervisor does not choose to share some identical pages, typically because it expects them to change too quickly. For RUBiS, the tracer under predicts slightly, probably because our memory tracer is unable to access all memory regions. SpecJBB obtains the smallest amount of sharing because it is the most memory intensive application, quickly filling the VM's memory with randomly generated data as the benchmark runs.

We compare two placement algorithms: our *sharing aware* approach attempts to place each virtual machine on the host that will maximize its page sharing and the *sharing oblivious* scheme does not consider sharing opportunities when placing virtual machines, and instead places each virtual machine on the first host it finds with sufficient spare capacity. Although the sharing oblivious approach does not explicitly utilize sharing information to guide placement, page sharing will still occur if it happens to place virtual machines together with common pages. In addition, this means that self-sharing within each VM occurs in both scenarios, so the improvements we see are caused by intelligent colocation leading to better inter-vm sharing. For simplicity, we assume that memory is the bottleneck resource and do not consider CPU or network bandwidth as a constraint.

Initially, we create one virtual machine of each type and place it on its own physical host. Additional VMs of each type are then spawned on a fifth host and migrated to one of the four primary hosts. We compare the number of virtual machines which can be successfully hosted using both our sharing aware algorithm which migrates each new VM to the host with the greatest sharing potential and a sharing oblivious placement algorithm which migrates each VM to the first host it finds with sufficient memory, without regard to sharing. The experiment terminates when no new virtual machines can be placed.

Each virtual machine is configured with 384 MB of RAM, and the hosts have 1.5 GB of spare memory since VMware reserves 0.5 GB for itself. Thus we expect each host to be able to run about four VMs without sharing. Figure 7 displays the final placements reached by each algorithm. The three web applications, TPC-W, OFBiz, and RUBiS, demonstrate a benefit from utilizing sharing, allowing more VMs to be packed than the base four. The sharing oblivious algorithm places four VMs on each host, except for host C on which it fits an extra VM due to the sharing between TPC-W instances. The sharing aware approach is able to place a total of 20 virtual machines, while the Oblivious approach can only fit 17. For this scenario, exploiting sharing increased the data center's capacity by a modest 17%.

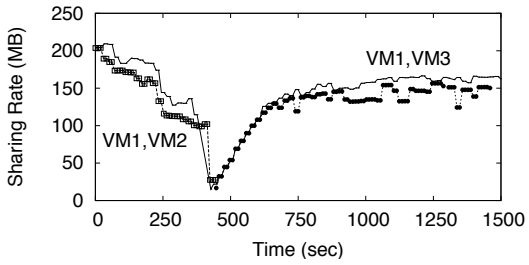


Figure 8. Hotspot mitigation: When a change in workload occurs, Memory Buddies migrates VMs to consolidate VMs with higher sharing potential on the same hosts.

We next use the memory traces gathered from these applications to simulate a larger data center. We increase the total number of hosts to 100 and measure the number of virtual machines which can be placed depending on whether sharing information is used. Our trace driven simulator utilizes the same control plane and algorithms as described previously. On this larger scale testbed, the sharing-aware approach places a total of 469 VMs, while the sharing oblivious approach can only host 406, giving a benefit of about 16% when using sharing. This matches well with the results from our testbed; the slight change in performance is due to the sharing oblivious approach getting "lucky" and placing more VMs together which happen to share pages.

Result: By reducing the total memory requirements on each host, the effective capacity of a data center can be increased. Our testbed and trace driven simulations obtain benefits of 16-17% due to increased inter-vm sharing when Memory Buddies guides VM placement.

7.4 Hotspot Mitigation

Workload variations occur over time for most web applications and this can reduce the potential for memory sharing between colocated VMs in data centers. We have reproduced a typical data center scenario to demonstrate Memory Buddies' ability to detect and respond to a memory hotspot when application phase changes. The experiment employs two hosts, the first running two virtual machines (VM_1 and VM_2) and the second running only one (VM_3). All of the virtual machines are allocated 512MB of memory and serve static files generated following the SPECweb99 specification [17] with Apache web servers. Initially, we use httperf [16] to send an identical set of requests to each server resulting in a high potential for sharing between VMs.

Figure 8 shows the amount of memory shared by each VM with the other VMs residing on the same host as reported by VMWare ESX. Since VM_1 and VM_2 are colocated, they initially have a high level of sharing at about 400MB. After 60 seconds of load injection, we trigger a phase change for the requests being sent to VM_2. As a result, the sharing between VM_1 and VM_2 decreases significantly putting more memory pressure on the host. This triggers Memory Buddies hotspot mitigation mechanism at time 360 seconds. Since VM_1 and VM_3 continue to receive the same workload, there is a high potential for sharing between them. Therefore Memory Buddies determines that VM_1 should be migrated to Host 2. After the migration completes, the sharing rate between VM_1 and VM_3 gradually increases again as ESX Server CBPS identifies sharable pages.

Result: Memory Buddies' monitoring system is able to detect changes in sharing potential brought on by application phase tran-

OS	CPU	RAM
Darwin 9.0.0	PowerBook 6, PowerPC	1152
Darwin 9.2.0	Macmini1, i386	1024
Darwin 9.4.0	MacBook2, i386	2048
Darwin 9.4.0	iMac7, i386	2048
Linux 2.6.9	Intel Family 15 Model 2	1010
Linux 2.6.18	Intel Family 6 Model 2	2018
Windows NT 5.1	x86 Family 6 Model 15	511

Table 2. Machine configurations (as reported by their operating system) considered in the desktop virtualization case study.

sitions. This type of hotspot is automatically resolved by determining a different host with a higher sharing potential for one of the VMs.

7.5 Case Study: Desktop Virtualization

Desktop virtualization consists of moving traditional desktop environments to virtual machines colocated in a data center. The user can then access his desktop environment using a thin-client interface from various locations. System administrators that are planning to deploy desktop virtualization need to estimate the memory requirements to host all the desktop VMs on the servers. The potential effects of intra and inter-VM memory sharing are not known a priori which makes it very hard to plan adequate memory resources. In this case study, we show how we can use our tools to answer "what if" questions in order to predict the memory requirements of colocated desktop virtual machines.

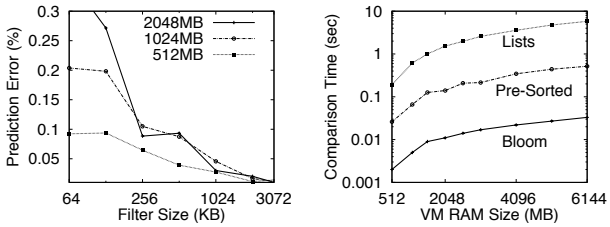
We have deployed our memory tracer on a set of real workstations running Windows, Linux and MacOS X on PowerPC and Intel platforms. We have collected memory traces from each machine every 30 minutes over several weeks. This data has been consolidated in a database to allow for easier mining. Table 2 summarizes the various desktop configurations we have considered.

By combining the traces of the different machines in the database, we can quickly compute how much memory sharing can be achieved for a particular combination of VMs. The number of unique hashes found in the combined traces represent the number of physical memory pages that will be needed by the hypervisor in case of perfect sharing. This upper-bound of sharing includes both inter and intra-VM sharing. We use the collected data and our tools to answer a number of "what if" questions with different combinations of OS colocation. Table 3 shows the results we obtained for 4 questions: what is the potential sharing (i) if 3 VMs have a different OS, (ii) if 2 VMs have the same OS but different versions and 1 VM has a different OS, (iii) if 3 VMs have the same OS but different versions and 1 VM has a different OS, (iv) all VMs have the same OS version but possibly different hardware platforms.

When heterogeneous OSes are combined (first line of the table), the sharing potential only comes from intra-VM sharing and remains at a modest 13%. When we replace Linux by another version of MacOS X, inter-VM sharing starts to play a significant role and memory sharing jumps to 35% overall. Adding different versions of the same operating system (Darwin 9.0, 9.2 and 9.4) maintains a substantial inter-VM sharing for an overall memory sharing close to 37%. When homogeneous software configurations are used even on different hardware platforms, we observe memory sharing up to 40%. These numbers represent the optimal memory sharing case and actual sharing might be lower depending on the hypervisor implementation of page sharing. Note that the predicted server memory does not account for the hypervisor memory requirements that are usually fixed and implementation dependent.

If the machine workload varies greatly over time, it is possible to perform these computations with different traces taken at different points in time to evaluate the memory sharing evolution over time. We found in our experiments that the predicted memory sharing did not change significantly over time for desktop machines. Computing the memory server prediction for a given configuration usually only takes few seconds but this may vary depending on the database size and number of traces to analyze. It is then possible to use the technique to answer a broad range of "what if" questions like sharing potential over time or in the presence of workload variations.

Result: Memory Buddies can be used offline to compute memory sharing and answer "what if" questions when planning for desktop virtualization. We found that colocating different OSes only uses intra-VM sharing. However mixing different versions of the same OS leads to substantial inter-VM sharing. As expected, the maximum sharing is observed when similar versions of an OS are colocated.



(a) Bloom Filter Estimation Error (b) Comparison Efficiency

Figure 9. Bloom filter accuracy vs efficiency tradeoff. Smaller Bloom filter bit vectors reduce the accuracy of sharing estimates, but also significantly reduce the computation time required for comparison.

VM RAM	Hash List Size (KB)	Bloom Size (KB) w/0.2% Error
1GB	1024	92
4GB	4096	124
8GB	8192	368

Table 4. Per VM communication cost in KB for hash lists and Bloom filters with a 0.2% error rate.

7.6 Fingerprint Efficiency and Accuracy

Memory Buddies allows a tradeoff between the accuracy, speed and space required for estimating sharing potential depending on whether hash lists or Bloom filters are used.

We first measure the accuracy of Bloom filter comparisons when varying the size of the Bloom filter's bit vector. We use pairs of traces gathered in our department study from systems with 512MB, 1GB, and 2GB of RAM. We report the average error in percent of total pages for four pairs of traces of each size. Figure 9(a) illustrates how the comparison error rapidly decreases as filter size rises, although larger memory sizes require bigger filters to prevent hash collisions. These results confirm the simulation data shown previously in Figure 3; a Bloom filter of only a few hundred KB is sufficient for an error rate of about 0.1%.

We next measure the time to compare two fingerprints to calculate the potential for sharing when using our exact and compact techniques. In both approaches, the computation time increases when using VMs with larger amounts of RAM, because either there are more hashes to be compared or a Bloom filter with a larger bit vector is required in order to meet a target accuracy level. Figure 9(b) demonstrates how the comparison time for a pair of VMs increases with memory size. The exact comparison technique using hash lists first sorts the two lists before comparing them. Since sort-

OS combination	Total memory	Shareable pages	Predicted server memory
Linux 2.6.9, Darwin 9.0.0, Windows NT 5.1	4223 MB	13.2%	3666 MB
Darwin 9.4.0, Darwin 9.0.0, Windows NT 5.1	5248 MB	35.3%	3397 MB
Darwin 9.2.0, Darwin 9.4.0, Darwin 9.0.0, Windows NT 5.1	6272 MB	36.8%	3966 MB
Darwin 9.4.0 (3 MacBook2 + 1 iMac7)	8192 MB	40.0%	4917 MB

Table 3. Server memory usage prediction for various colocation configuration. Total memory represents the total memory required without sharing and predicted memory is the required memory on the server when all possible sharable pages are actually shared.

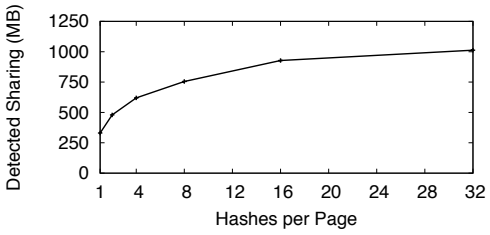


Figure 10. Splitting each page into multiple chunks allows Memory Buddies to detect sub-page level sharing between similar pages.

ing can be the dominant cost, we also present the time when lists are presorted by each VM prior to sending the hash lists to the control plane. Presorting the lists decreases the comparison time by about an order of magnitude, but incurs overhead on each host in the system. Switching to Bloom filters reduces the time further, but at the expense of reduced accuracy.

The total communication overhead of the system is dependent on the number of VMs running in the data center, the amount of RAM used by each VM, and the fingerprinting method used. Table 4 compares the cost of storing or transmitting Bloom filter based memory fingerprints or hash lists of various sizes. Fingerprints only need to be gathered once every few minutes, incurring minimal network cost if there is a small number of VMs. For very large data centers, the overhead of transmitting full hash lists can become prohibitive, while the Bloom filter approach remains manageable.

Result: Employing Bloom filters in large data centers can reduce sharing estimation time by an order of magnitude and can reduce network overheads by over 90%, while still maintaining a high degree of accuracy.

7.7 Sub-Page Sharing

While VMware ESX currently only supports memory sharing at the granularity of full pages, recent research has demonstrated that significant benefits can be obtained by sharing portions of similar, but not identical pages [8]. We have added preliminary support to Memory Buddies for detecting sub-page level sharing between systems by breaking each page into a series of n chunks, each of which is mapped to a 32bit hash. As a result, Memory Buddies produces a fingerprint n times as large for each system, but it can use its existing fingerprint comparison tools to detect similarity between different VMs.

To demonstrate the benefits of sub-page sharing, we have analyzed the amount of sharing achieved between two systems running 64bit Ubuntu Linux, each with 2GB of RAM, when the number of hashes per page is varied between one and thirty two. Figure 10 illustrates how subpage level sharing can triple the total amount of sharable memory. The number of hashes per page could be selected by the system operator to balance the added overhead of larger fingerprints against the increased accuracy in sub-page level sharing estimation.

Result: Although Memory Buddies does not currently use a hypervisor that supports sub-page level sharing, it can efficiently detect similar pages by generating multiple hashes per page. This can provide significant benefits in total sharing.

8. Related Work

Transparent page sharing in a virtual machine hypervisor was implemented in the Disco system [3]; however it required guest operating system modification, and detected identical pages based on factors such as origin from the same location on disk. Content-based page sharing was introduced in VMware ESX [27], and later in Xen [11]. These implementations use background hashing and page comparison in the hypervisor to transparently identify identical pages, regardless of their origin. Since our prototype lacks access to the memory hashes gathered by the hypervisor, we duplicate this functionality in the guest OS. In Memory Buddies, however, we extend the use of these page content hashes in order to detect the potential for memory sharing between distinct physical hosts, rather than within a single host.

The Difference Engine system was recently proposed as a means to enable even higher degrees of page sharing by allowing portions of similar pages to be shared [8]. Although Memory Buddies has preliminary support for detecting sub-page sharing across machines by using multiple hashes per page, it currently relies on ESX's sharing functions which do not support sub-page level sharing. We believe that as the technologies to share memory become more effective and efficient, the benefits of using page sharing to guide placement will continue to rise.

Process migration was first investigated in the 80's [19; 26]. The re-emergence of virtualization led to techniques for virtual machine migration performed over long time scales in [22; 28; 12]. The means for "live" migration of virtual machines incurring downtimes of only tens of milliseconds have been implemented in both Xen [5] and VMware [18]. At the time of writing, however, only VMware ESX server supports both live migration and page sharing simultaneously.

Virtual machine migration was used for dynamic resource allocation over large time scales in [21; 25; 6]. Previous work [31] and the VMware Distributed Resource Scheduler [29] monitor CPU, network, and memory utilization in clusters of virtual machines and use migration for load balancing. The Memory Buddies system is designed to work in conjunction with these multi-resource load balancing systems by providing a means to use page sharing to help guide placement decisions. Moreover, offline planning of memory resources for desktop virtualization can be predicted accurately rather than relying on generic rules of thumb that are recommended by manufacturers.

Bloom filters were first proposed in [1] to provide a tradeoff between space and accuracy when storing hash coded information. Guo et al. provide a good overview of Bloom filters as well as an introduction to intersection techniques [7]. Bloom filters have also been used to rapidly compare search document sets in [10] by comparing the inner product of pairs of Bloom filters. The Bloom filter intersection technique we use provides a more accurate estimate based on the Bloom filter properties related to the probability of individual bits being set in the bit vector. This approach was used in [15] to detect similar workloads in peer to peer networks.

9. Conclusions

Modern hypervisors implement a technique called content-based page sharing (CBPS) that maps duplicate copies of a page resident on a host onto a single physical memory page. In this paper, we have presented Memory Buddies, a system that provides sharing-aware colocation of virtual machines by optimizing CBPS usage by consolidating VMs with higher sharing potential on the same host.

We have made three contributions: (i) a fingerprinting technique —based on hash lists or Bloom filters— to capture VM memory content and identify high page sharing potential, (ii) a smart VM colocation algorithm that can be used for both initial placement of virtual machines or to consolidate live environments and adapt to load variations using a hotspot mitigation algorithm, and (iii) a collection of memory traces of real-world systems that we are making available to other researchers to validate and explore further memory sharing experiments.

Using a mix of enterprise and ecommerce applications, we showed that our Memory Buddies system is able to increase the effective capacity of a data center by 17% when consolidating VMs with higher page sharing potential. These gains come from intelligently grouping VMs with similar memory images to encourage a high degree of inter-vm page sharing. We also showed that our system can effectively detect and resolve memory hotspots due to changes in sharing patterns. Our tools can also be used for capacity planning in scenarios such as desktop virtualization. Memory Buddies can be easily integrated in existing hypervisors such as VMWare ESX server and their management infrastructure to optimize the placement and migration of VMs in data centers.

As future work, we are continuing to collect memory traces from desktops and servers to perform a more extensive trace analysis. We also plan to enhance our techniques to dynamically vary how much memory is allocated to VMs to improve memory utilization and further increase the number of VMs that can be placed on a data center while guaranteeing application performance.

Acknowledgments

We are grateful for the contributions of Emery Berger and James Cipar who took part in the initial project discussions and the first generation of our memory tracer software. Timur Alperovich worked on the Macintosh and Linux versions of the tracer, and Michael Sindelar helped with some initial analysis of the gathered traces. Finally we would like to thank the members of our department who volunteered to run the memory tracer on their machine.

This work was supported in part by NSF grants CNS-0720271 and CNS-0720616. Mark Corner was supported by CNS-0447877.

References

[1] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[2] Andrei Broder and Michael Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003.

[3] Edouard Bugnion, Scott Devine, and Mendel Rosenblum. DISCO: Running Commodity Operating Systems on Scalable Multiprocessors. In *SOSP*, pages 143–156, 1997.

[4] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In *4th ACM/IFIP/USENIX International Middleware Conference*, June 2003.

[5] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfiel. Live migration of virtual machines. In *Proceedings of Usenix Symposium on Network Systems Design and Implementation (NSDI)*, May 2005.

[6] Laura Grit, David Irwin, , Aydan Yumerefendi, and Jeff Chase. Virtual machine hosting for networked clusters: Building the foundations for autonomic orchestration. In *Workshop on Virtualization Technology in Distributed Computing (VTDC)*, November 2006.

[7] Deke Guo, Jie Wu, Honghui Chen, and Xueshan Luo. Theory and Network Applications of Dynamic Bloom Filters. In *INFOCOM*, 2006.

[8] Diwaker Gupta, Sangmin Lee, Michael Vrable, Stefan Savage, Alex C. Snoeren, George Varghese, Geoffrey M. Voelker, and Amin Vahdat. Difference engine: Harnessing memory redundancy in virtual machines. In *Usenix OSDI*, December 2008.

[9] Paul Hsieh. *Hash functions*. http://www.azillionmonkeys.com/qed/hash.html.

[10] Navendu Jain, Michael Dahlin, and Renu Tewari. Using Bloom Filters to Refine Web Search Results. In *WebDB*, pages 25–30, 2005.

[11] Jacob Kloster, Jesper Kristensen, and Arne 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.

[12] M. Kozuch and M. Satyanarayanan. Internet suspend and resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, Calicoon, NY*, June 2002.

[13] Purushottam Kulkarni, Prashant J. Shenoy, and Weibo Gong. Scalable Techniques for Memory-efficients CDN Simulations. In *WWW*, 2003.

[14] libvirt. *The Virtualization API*. http://libvirt.org.

[15] Xucheng Luo, Zhiguang Qin, Ji Geng, and Jiaqing Luo. IAC: Interest-Aware Caching for Unstructured P2P. In *SKG*, page 58, 2006.

[16] David Mosberger and Tai Jin. httperf: A tool for measuring web server performance. In *First Workshop on Internet Server Performance*, pages 59—67. ACM, June 1998.

[17] E. Nahum. Deconstructing specweb, 2002.

[18] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast Transparent Migration for Virtual Machines. In *USENIX Annual Technical Conference*, 2005.

[19] M. Powell and B. Miller. Process migration in DEMOS/MP. *Operating Systems Review*, 17(5):110–119, 1983.

[20] Apache Open For Business Project. http://ofbiz.apache.org.

[21] Paul Ruth, Junghwan Rhee, Dongyan Xu, Rick Kennell, and Sebastien Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In *IEEE International Conference on Autonomic Computing (ICAC)*, June 2006.

[22] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[23] W. Smith. TPC-W: Benchmarking An Ecommerce Solution. `http://www.tpc.org/information/other/techarticles.asp`.

[24] The standard performance evaluation corporation (spec). `http://www.spec.org`.

[25] A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance in Virtual Environments through Run-time Inference and Adaptation. In *Fourteenth International Symposium on High Performance Distributed Computing (HPDC)*, July 2005.

[26] M. M. Theimer, K. A. L., and D. R. Cheriton. Preemptable Remote Execution Facilities for the V-System. pages 2–12, December 1985.

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

[28] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[29] VMware Whitepaper. Drs performance and best practices.

[30] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *International Middleware Conference*, 2008.

[31] Timothy Wood, Prashant J. Shenoy, Arun Venkataramani, and Mazin S. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *Networked Systems Design and Implementation (NSDI '07)*, 2007.