DIWAKER GUPTA, SANGMIN LEE, MICHAEL VRABLE, STEFAN SAVAGE, ALEX C. SNOEREN, GEORGE VARGHESE, GEOFFREY M. VOELKER, AND AMIN VAHDAT

Difference Engine

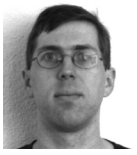
VIRTUALIZATION TECHNOLOGY HAS improved dramatically over the past decade and has now become pervasive within the service-delivery industry. Virtual machines are particularly attractive for server consolidation. Their strong resource and fault-isolation guarantees allow multiplexing of hardware among individual services, each configured with a custom operating system. Although physical CPUs are frequently amenable to multiplexing, main memory is not. Thus, memory is often the primary bottleneck to increasing the degree of multiplexing in enterprise and data center settings. Difference Engine [1] enables virtual machine (VM) monitors to allocate more machine memory for VMs than is present in the system, by using aggressive memory sharing techniques. As with VMware ESX server, Difference Engine shares identical memory pages. In addition, Difference Engine also shares pages with only partial content overlap and compresses infrequently used pages, enabling it to further improve memory savings by up to a factor of 2.5 compared to identical page sharing alone in VMware ESX server.

Diwaker Gupta's Ph.D research focused on virtualization, network emulation, and large-scale system testing. He is also interested in cloud computing and Web applications. He is currently employed at Aster Data Systems.

diwaker@asterdata.com

Sangmin Lee is a Ph.D student in the Department of Computer Sciences at the University of Texas, Austin. His research interests include distributed computing and operating systems.

sangmin@cs.utexas.edu



Michael Vrable is pursuing a Ph.D. in computer science at the University of California, San Diego, and is advised by professors Stefan Savage and Geoffrey Voelker. He received an M.S. from UCSD and a B.S. from Harvey Mudd College.

mvrable@cs.ucsd.edu

Stefan Savage is an associate professor of computer science at the University of California, San Diego. He has a B.S. in history and reminds his colleagues of this fact any time the technical issues get too complicated.

savage@cs.ucsd.edu

Alex C. Snoeren is an associate professor in the Computer Science and Engineering Department at the University of California, San Diego. His research interests include operating systems, distributed computing, and mobile and wide-area networking.

snoeren@cs.ucsd.edu

George Varghese is a professor of computer science at the University of California, San Diego. Several algorithms he has helped develop have found their way into commercial systems including Linux (timing wheels), the Cisco GSR (DRR), and Microsoft Windows (IP lookups).

varghese@cs.ucsd.edu

Geoffrey M. Voelker is an associate professor of computer science and engineering at the University of California, San Diego. He works in computer systems and networking.

voelker@cs.ucsd.edu

Amin Vahdat is a professor at the University of California, San Diego. His research focuses broadly on computer systems, including distributed systems, networks, and operating systems.

vahdat@cs.ucsd.edu

With main memory as a consolidation bottleneck, researchers and commercial VM software vendors have developed techniques to decrease the memory requirements for virtual machines. The VMware ESX server implements content-based page sharing, in which virtual pages in different VMs have identical content and therefore can share the same machine page copy-on-write. Identical page sharing has been shown to reduce the memory footprint of multiple, homogeneous virtual machines by 10%–40% [2]. We found, however, that the benefits of identical page sharing decline rapidly when more heterogeneous guest VMs are used.

The premise of this work is that there are significant additional benefits from sharing at a sub-page granularity (i.e., there are many pages that are *nearly* identical). We show that it is possible to efficiently find such similar pages and to coalesce them into a much smaller memory footprint. Among the set of similar pages, we are able to store most as patches relative to a single baseline page.

We also compress those pages that are unlikely to be accessed in the near future. In both patching and compression, Difference Engine relies on finding pages that are less frequently used to offset the cost of recovering these pages. To support these techniques, we added a swapping service so that even when memory has been oversubscribed (by allocating more memory than exists), all VM guests will have access to the memory their OS was configured to use by leveraging disk as secondary storage.

Difference Engine provides these benefits without negatively impacting application performance: in our experiments across a variety of workloads, Difference Engine imposes less than 7% execution time overhead. In return, we further show that Difference Engine can take advantage of the improved memory efficiency to increase aggregate system performance by utilizing the free memory to create additional virtual machines in support of a target workload. Thus, for a prototypical Internet service workload, Difference Engine is able to use the additional memory to increase maximum request throughput by nearly 40%.

Architecture

Difference Engine uses three distinct mechanisms that work together to realize the benefits of memory sharing, as shown in Figure 1. In this example, two VMs have allocated five pages total, each initially backed by distinct pages in machine memory (Figure 1a). For brevity, we only show how the mapping from guest physical memory to machine memory changes; the guest virtual to guest physical mapping remains unaffected. First, for identical pages across the VMs, we store a single copy and create references that point to the original. In Figure 1b, one page in VM-2 is identical to one in VM-1. For pages that are similar but not identical, we store a

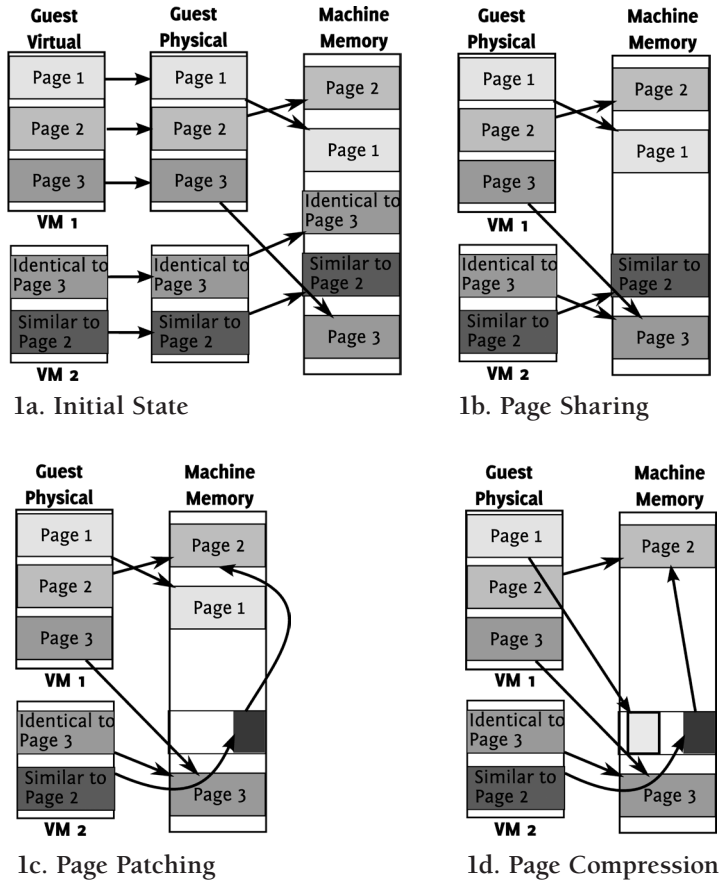


FIGURE 1: THE INITIAL STATE AND THE THREE DIFFERENT MEMORY CONSERVATION TECHNIQUES EMPLOYED BY DIFFERENCE ENGINE: PAGE SHARING, PAGE PATCHING, AND COMPRESSION. IN THIS EXAMPLE, FIVE PHYSICAL PAGES ARE STORED IN LESS THAN THREE MACHINE MEMORY PAGES FOR A SAVINGS OF ROUGHLY 50%.

patch against a reference page and discard the redundant copy. In Figure 1c, the second page of VM-2 is stored as a patch to the second page of VM-1. Finally, for pages that are unique and infrequently accessed, we compress them in memory to save space. In Figure 1d, the remaining private page in VM-1 is compressed. The actual machine memory footprint is now less than three pages, down from five pages originally.

In all three cases, efficiency concerns require us to select candidate pages that are unlikely to be accessed in the near future. We employ a global clock that scans memory in the background, identifying pages that have not been recently used. In addition, reference pages for sharing or patching must be found quickly without introducing performance overhead. Difference Engine uses full-page hashes and hash-based fingerprints to identify good candidates. Finally, we implement a demand paging mechanism that supplements main memory by writing VM pages to disk to support overcommitment, allowing the total memory required for all VMs to temporarily exceed the machine memory capacity.

Page Sharing

Difference Engine's implementation of content-based page sharing is similar to those in earlier systems. We walk through memory looking for identical pages. As we scan memory, we hash each page and index it based on its hash value. Identical pages hash to the same value and a collision indicates that a potential matching page has been found. We perform a byte-by-byte comparison to ensure that the pages are indeed identical before sharing them.

Upon identifying target pages for sharing, we reclaim one of the pages and update the virtual memory to point at the shared copy. Both mappings are marked read-only, so that writes to a shared page cause a page fault that will be trapped by the virtual machine monitor (VMM). The VMM returns a private copy of the shared page to the faulting VM and updates the virtual memory mappings appropriately. If no VM refers to a shared page, the VMM reclaims it and returns it to the free memory pool.

Patching

Traditionally, the goal of page sharing has been to eliminate redundant copies of identical pages. Difference Engine considers further reducing the memory required to store similar pages by constructing patches that represent a page as the difference relative to a reference page.

One of the principal complications with subpage sharing is identifying candidate reference pages. Difference Engine uses a parametrized scheme to identify similar pages based upon the hashes of several 64-byte portions of each page. In particular, HashSimilarityDetector(k,s) hashes the contents of ($k \times s$) 64-byte blocks at randomly chosen locations on the page and then groups these hashes together into k groups of s hashes each. We use each group as an index into a hash table.

Higher values of s capture local similarity, whereas higher k values incorporate global similarity. Hence, HashSimilarityDetector(1,1) will choose one block on a page and index that block; pages are considered similar if that block of data matches. HashSimilarityDetector(1,2) combines the hashes from two different locations in the page into one index of length two. HashSimilarityDetector(2,1) instead indexes each page twice: once based on the contents of a first block, and again based on the contents of a second block.

Pages that match at least one of the two blocks are chosen as candidates. Through experimentation, we discovered that HashSimilarityDetector(2,1) with one candidate does surprisingly well. There is a substantial gain from hashing two distinct blocks in the page separately, but little additional gain by hashing more blocks.

Difference Engine indexes a page by hashing 64-byte blocks at two fixed locations in the page (chosen at random) and uses each hash value as a separate index to store the page in the hash table. To find a candidate similar page, the system computes hashes at the same two locations, looks up those hash table entries, and calculates the page patch to determine memory savings if it finds a match for either of the indexed blocks.

Our current implementation uses 18-bit hashes to keep the hash table small to cope with the limited size of the Xen heap. In general, though, larger hashes might be used for improved savings and fewer collisions. Our analysis suggests, however, that the benefits from increasing the hash size are modest.

Compression

Finally, for pages that are not significantly similar to other pages in memory, we consider compressing them to reduce the memory footprint. Compression is useful only if the compression ratio is reasonably high and, like patching, if selected pages are accessed infrequently. Otherwise, the overhead of compression/decompression will outweigh the benefits. We identify candidate pages for compression using a global clock algorithm (see "Clock," below), assuming that pages that have not been recently accessed are unlikely to be accessed in the near future.

Difference Engine supports multiple compression algorithms, currently LZO and WKdm as described in Wilson et al. [3]; we invalidate compressed pages in the VM and save them in a dynamically allocated storage area in machine memory. When a VM accesses a compressed page, Difference Engine decompresses the page and returns it to the VM uncompressed. It remains there until it is again considered for compression.

Paging Machine Memory

Although Difference Engine will deliver some (typically high) level of memory savings, in the worst case all VMs might actually require all of their allocated memory. Setting aside sufficient physical memory to account for this prevents Difference Engine from using the memory to create additional VMs. Not doing so, however, may result in temporarily overshooting the physical memory capacity of the machine and causing a system crash. We therefore require a demand-paging mechanism to supplement main memory by writing pages out to disk in such cases.

A good candidate page for swapping out should not be accessed in the near future—the same requirement as compressed/patched pages. In fact, Difference Engine also considers compressed and patched pages as candidates for swapping out. Once the contents of the page are written to disk, the page can be reclaimed. When a VM accesses a swapped-out page, Difference Engine fetches it from disk and copies the contents into a newly allocated page that is mapped appropriately in the VM's memory.

Since disk I/O is involved, swapping in/out is an expensive operation. Further, a swapped page is unavailable for sharing or as a reference page for patching. Therefore, swapping should be an infrequent operation. Difference

Engine implements the core mechanisms for paging and leaves policy decisions, such as when and how much to swap, to user-level tools.

Implementation

We have implemented Difference Engine in the Xen 3.0.4 VMM in roughly 14,500 lines of code. An additional 20,000 lines come from ports of existing patching and compression algorithms (Xdelta, LZO, WKdm) to run inside Xen.

Xen and other platforms that support fully virtualized guests use a mechanism called "shadow page tables" to manage guest OS memory [2]. The guest OS has its own copy of the page table that it manages, believing that they are the hardware page tables, though in reality they are just a map from the guest's virtual memory to its notion of physical memory (V2P map). In addition, Xen maintains a map from the guest's notion of physical memory to the machine memory (P2M map). The shadow page table is a cache of the results of composing the V2P map with the P2M map, mapping guest virtual memory directly to machine memory.

Difference Engine relies on manipulating P2M maps and the shadow page tables to interpose on page accesses. For simplicity, we do not consider any pages mapped by Domain-0 (the privileged, control domain in Xen), which, among other things, avoids the potential for circular page faults.

Clock

Difference Engine implements a not-recently-used (NRU) policy [4] to select candidate pages for sharing, patching, compression, and swapping out. On each invocation, the clock scans a portion of machine memory, checking and clearing the referenced (R) and modified (M) bits on pages. Thus, pages with the R or the M bit set must have been referenced or modified since the last scan. We ensure that successive scans of memory are separated by at least four seconds in the current implementation, to give domains a chance to set the R/M bits on frequently accessed pages. In the presence of multiple VMs, the clock scans a small portion of each VM's memory in turn for fairness. The external API exported by the clock is simple: Return a list of pages (of some maximum size) that have not been accessed in some time.

In OSes running on bare metal, the R/M bits on page-table entries are typically updated by the processor. Xen structures the P2M map exactly like the page tables used by the hardware. However, since the processor does not actually use the P2M map as a page table, the R/M bits are not updated automatically. We modify Xen's shadow page table code to set these bits when creating readable or writable page mappings. Unlike conventional operating systems, where there may be multiple sets of page tables that refer to the same set of pages, in Xen there is only one P2M map per domain. Hence, each guest page corresponds unambiguously to one P2M entry and one set of R/M bits.

Real-World Applications

We now present the performance of Difference Engine on a variety of workloads. We seek to answer two questions. First, how effective are the memory-saving mechanisms at reducing memory usage for real-world applications? Second, what is the impact of those memory-sharing mechanisms on system performance? Since the degree of possible sharing depends on

the software configuration, we consider several different cases of application mixes.

To put our numbers in perspective, we conduct head-to-head comparisons with VMware ESX Server for three different workload mixes. We run ESX Server 3.0.1 build 32039 on a Dell PowerEdge 1950 system. Note that even though this system has two 2.3-GHz Intel Xeon processors, our VMware license limits our usage to a single CPU. We therefore restrict Xen (hence, Difference Engine) to use a single CPU for fairness. We also ensure that the OS images used with ESX match those used with Xen, especially the file system and disk layout. Note that we are only concerned with the effectiveness of the memory sharing mechanism, not in comparing the application performance across the two hypervisors. Further, we configure ESX to use its most aggressive page sharing settings, in which it scans 10,000 pages/second (compared to its default of 200); we configure Difference Engine similarly.

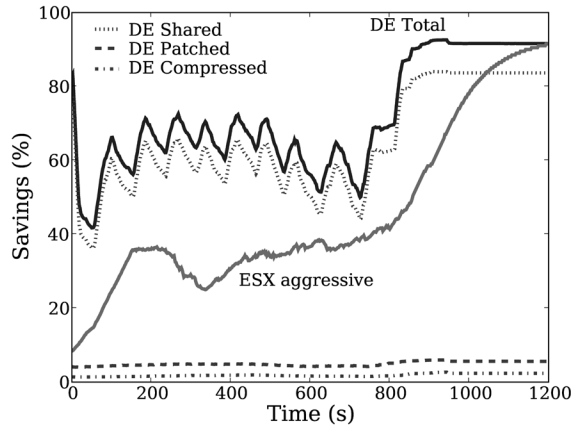


FIGURE 2: FOUR IDENTICAL VMS EXECUTE DBENCH. FOR SUCH HOMOGENEOUS WORKLOADS, BOTH DIFFERENCE ENGINE AND ESX EVENTUALLY YIELD SIMILAR SAVINGS, BUT DE EXTRACTS MORE SAVINGS WHILE THE BENCHMARK IS IN PROGRESS.

In our first set of benchmarks, we test the base scenario where all VMs on a machine run the same OS and applications. This scenario is common in cluster-based systems where several services are replicated to provide fault tolerance or load balancing. Our expectation is that significant memory savings are available and that most of the savings will come from page sharing. The graphs shown in Figures 2–4 break out the contributions in Difference Engine by page compression (the least), patching, and page sharing (the most) against page sharing in ESX Server.

We set up four 512-MB virtual machines running Debian 3.1. Each VM executes dbench for 10 minutes followed by a stabilization period of 20 minutes. Figure 2 shows the amount of memory saved as a function of time. First, note that eventually both ESX and Difference Engine reclaim roughly the same amount of memory (with the graph for ESX plateauing beyond 1,200 seconds). However, while dbench is executing, Difference Engine delivers approximately 1.5 times the memory savings achieved by ESX. As before, the bulk of Difference Engine savings comes from page sharing for the homogeneous workload case.

We used two different sets of guests VMs for testing heterogeneous performance.

- MIXED-1: Windows XP SP1 hosting RUBiS; Debian 3.1 compiling the Linux kernel; Slackware 10.2 compiling Vim 7.0 followed by a run of the lmbench benchmark.

- MIXED-2: Windows XP SP1 running Apache 2.2.8 hosting approximately 32,000 static Web pages crawled from Wikipedia, with httperf running on a separate machine requesting these pages; Debian 3.1 running the SysBench database benchmark using 10 threads to issue 100,000 requests; Slackware 10.2 running dbench with 10 clients for six minutes followed by a run of the IOZone benchmark.

Figures 3 and 4 show the memory savings as a function of time for the two heterogeneous workloads, MIXED-1 and MIXED-2. We make the following observations. First, in steady state, Difference Engine delivers a factor of 1.6 to 2.5 more memory savings than ESX. For instance, for the MIXED-2 workload, Difference Engine could host the three VMs allocated 512 MB of physical memory each in approximately 760 MB of machine memory; ESX would require roughly 1100 MB of machine memory. The remaining, significant, savings come from patching and compression. And these savings come at a small cost. The baseline configuration is regular Xen without Difference Engine. In all cases, performance overhead of Difference Engine is within 7% of the baseline. For the same workload, we find that performance under ESX with aggressive page sharing is also within 5% of the ESX baseline with no page sharing.

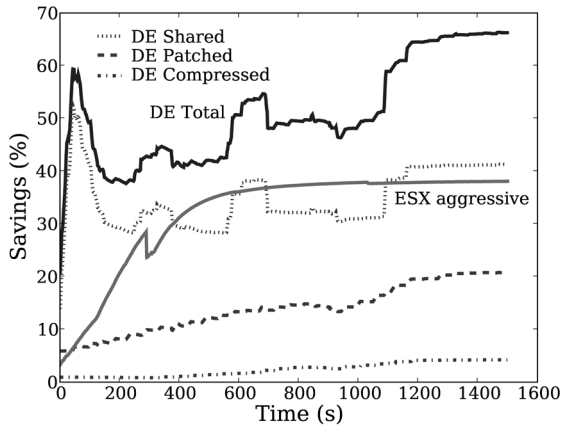


FIGURE 3: MEMORY SAVINGS FOR MIXED-1. DIFFERENCE ENGINE SAVES UP TO 45% MORE MEMORY THAN ESX.

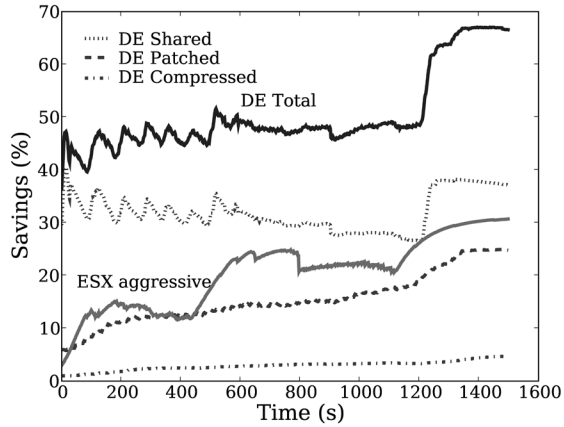


FIGURE 4: MEMORY SAVINGS FOR MIXED-2. DIFFERENCE ENGINE SAVES ALMOST TWICE AS MUCH MEMORY AS ESX.

Conclusion

One of the primary bottlenecks to higher degrees of virtual machine multiplexing is main memory. Earlier work shows that substantial memory savings are available from harvesting identical pages across virtual machines when running homogeneous workloads. The premise of this work is that there are significant additional memory savings available from locating and patching similar pages and in-memory page compression. We present the design and evaluation of Difference Engine to demonstrate the potential memory savings available from leveraging a combination of whole page sharing, page patching, and compression. We discuss our experience addressing a number of technical challenges, including algorithms to quickly identify candidate pages for patching, demand paging to support oversubscription of total assigned physical memory, and a clock mechanism to identify appropriate target machine pages for sharing, patching, compression, and paging. Our performance evaluation shows that Difference Engine delivers an additional factor of 1.6 to 2.5 more memory savings than VMware ESX Server for a variety of workloads, with minimal performance overhead. Difference Engine mechanisms might also be used to improve single OS memory management; we leave such exploration to future work.

ACKNOWLEDGMENTS

We would like to particularly thank Rik Farrow for crafting a condensed draft of this article from our conference paper. In the course of the project, we also received invaluable assistance from a number of people at VMware. We would like to thank Carl Waldspurger, Jennifer Anderson, and Hemant Gaidhani, and the Performance Benchmark group for feedback and discussions on the performance of ESX server. Also, special thanks are owed to Kiran Tati for assisting with ESX setup and monitoring and to Emil Sit for providing insightful feedback on the paper. Finally, we would like to thank Michael Mitzenmacher for his assistance with min-wise hashing, our shepherd Fred Douglis for his insightful feedback and support, and the anonymous OSDI '08 reviewers for their valuable comments. This work was supported in part by NSF CSR-PDOS Grant No. CNS-0615392, the UCSD Center for Networked Systems (CNS), and UC Discovery Grant 07-10237. Vrable was supported in part by an NSF Graduate Research Fellowship.

REFERENCES

[1] 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," *Proceedings of OSDI '08*: http://www.usenix.org/events/osdi08/tech/full_papers/gupta/gupta_html/.

[2] C.A. Waldspurger, "Memory Resource Management in VMware ESX Server," *Proceedings of OSDI '02*: http://www.usenix.org/publications/library/proceedings/osdi02/tech/waldspurger.html.

[3] P.R. Wilson, S.F. Kaplan, and Y. Smaragdakis, "The Case for Compressed Caching in Virtual Memory Systems," *Proceedings of the 1999 USENIX Annual Technical Conference*: http://www.usenix.org/publications/library/proceedings/usenix99/full_papers/wilson/wilson_html/.

[4] A.S. Tanenbaum, *Modern Operating Systems* (Englewood Cliffs, NJ: Prentice Hall, 2007).