Disaggregated Memory Benefits for Server Consolidation

Kevin Lim, Yoshio Turner, Jichuan Chang, Jose Renato Santos, Parthasarathy Ranganathan

HP Laboratories
HPL-2011-31

Keyword(s):

disaggregated memory, virtual machines, consolidation, hypervisor

Abstract:

Recent architecture research has introduced a new building block - a memory blade - which provides disaggregated memory capacity expansion and sharing for an ensemble of blade servers. In this paper, we examine the systems implications of this new architectural building block. We build a software-based prototype of memory disaggregation and examine how the additional level of indirection provided by the memory blade can provide significantly higher levels of consolidation. We specifically examine the use case of multiple large memory virtual machines (VMs) consolidated onto a single server. We explore content based sharing strategies to maximize the utilization of both local and remote memory. Using an in-memory database workload, our results show significantly higher levels of consolidation versus baseline servers (twice as many VMs than a memory-constrained baseline, providing 47% higher throughput), cost-effective memory expansion (28% better performance-per-dollar versus a large memory baseline), and effective content based sharing (up to 40%).

External Posting Date: March 6, 2011 [Fulltext] Approved for External Publication
Internal Posting Date: March 6, 2011 [Fulltext]

Disaggregated Memory Benefits for Server Consolidation

Kevin Lim Yoshio Turner Jichuan Chang Jose Renato Santos Parthasarathy Ranganathan
HP Labs

Abstract

Recent architecture research has introduced a new building block – a memory blade – which provides *disaggregated* memory capacity expansion and sharing for an ensemble of blade servers. In this paper, we examine the systems implications of this new architectural building block. We build a software-based prototype of memory disaggregation and examine how the additional level of indirection provided by the memory blade can provide significantly higher levels of consolidation. We specifically examine the use case of multiple large-memory virtual machines (VMs) consolidated onto a single server. We explore content based sharing strategies to maximize the utilization of both local and remote memory. Using an in-memory database workload, our results show significantly higher levels of consolidation versus baseline servers (twice as many VMs than a memory-constrained baseline, providing 47% higher throughput), cost-effective memory expansion (28% better performance-per-dollar versus a large memory baseline), and effective content based sharing (up to 40%).

1 Introduction

Recent research introduced a disaggregated memory architecture, which adds a level to the memory hierarchy in the form of expanded remote memory capacity that can be dynamically partitioned among multiple servers in a blade enclosure [3]. Remote memory is provided through a new building block, a memory blade, which consists of a large pool of commodity memory, an ASIC memory controller, and interface logic for communication between the memory blade and compute blades. Disaggregated memory is the key to enabling *independent scaling* of compute and memory resources, allowing compute servers to access enough memory capacity to match multi- or many-core processors, run large memory applications, and handle consolidated virtual machine environments. Previous work [3] presented microarchitecture simulation results indicating that disaggregated memory is a promising approach for a wide range of large-memory application workloads in data centers.

This paper significantly extends the previous work in three main ways. First, it presents the design and evaluation of system software support for disaggregated memory. The design leverages indirection at the hypervisor level to transparently integrate remote memory capacity into the physical address space of each guest operating system (OS) instance. This allows legacy server workloads to take advantage of the large remote memory capacity without changes at kernel-level or application-level. Furthermore, the design provides this support using only modest extensions to the hypervisor memory management subsystem and with no reduction in the performance of local memory accesses. Our evaluation on a prototype system confirms the expected benefits of disaggregated memory and finds that the relatively low latency of remote memory compared to traditional paging devices shifts the design trade-off for page replacement schemes in favor of fast victim selection over minimizing remote memory accesses.

Second, this paper studies the use of disaggregated memory to improve data center server consolidation using virtual machines (VMs). Server consolidation is accomplished by running multiple VM-based services on each physical server, reducing the number and total cost of physical servers required to support a given data center workload. Server consolidation is playing a growing role as data center operators seek to minimize costs, and as compute capacity rapidly increases in the many-core processor era. In order to run services with minimal slowdown compared to dedicated servers, co-located VMs must share the resources of a physical server without heavy contention. While CPU scheduling techniques can be used to dynamically multiplex CPU usage across VMs to closely match VM compute requirements over time, it is more challenging to efficiently share memory capac-

ity. Each VM has a pre-set expectation of the physical memory capacity it can access. Techniques like ballooning and demand-paging to storage media can be used in some cases to multiplex physical memory usage by VMs, but in general each VM can require high performance from all of its physical memory, precluding the effectiveness of these techniques. In practice, this means that a physical server's memory capacity often needs to be assigned conservatively to VMs. Without allowing memory overcommitment, memory capacity easily becomes the bottleneck resource that limits the achievable level of server consolidation. Disaggregated memory can potentially relieve that bottleneck by introducing a memory hierarchy layer with large capacity and main memory-like performance. Our study evaluates the real-world impact of disaggregated memory on server consolidation on a prototype Xen hypervisor-based system exhibiting complex interactions of hypervisor, guests and resources not modeled in previous simulation-based studies.

Finally, this paper unifies system software support for disaggregated memory with content-based page sharing (CBPS), a previous hypervisor technique that improves the efficient use of memory capacity by transparently copy-on-write sharing memory pages that have identical content [5]. Our evaluation shows that the combination of these two techniques further improves server consolidation.

2 Review of Hardware Architecture

In a disaggregated memory design, the memory hierarchy is expanded to include a remote level provided by separate memory blades. Disaggregated memory thus breaks the typical co-location of compute and memory to enable an architecture that allows for independent scaling of compute and memory capacity. Whereas commodity servers are limited in their memory capacity by the memory performance requirements and technology scaling trends, memory blades are specifically designed to provide large capacity through the use of buffer-on-board, or other fan-out techniques. By providing this capacity to multiple servers, the cost associated with such technologies is effectively amortized across the servers.

A server's local and remote memory together constitutes its machine address space. An application's locality of memory reference utilizes the server's local memory to maintain high performance, while remote memory provides memory size expansion at the cost of longer access latency (albeit still orders of magnitude faster than accessing today's persistent storage). In our designs, we utilize a PCI Express (PCIe) interconnect between the servers and the memory blades. Because the access time of remote data is dominated by transfer latency over the interconnect, lowering remote DRAM speed has negligible performance impact, providing hardware designers the luxury to trade off speed for increased capacity and power efficiency. In essence, disaggregated memory maintains high performance using fast local memory, while addressing the capacity and power concerns using slower remote memory, simultaneously satisfying all the key requirements of a modern memory system.

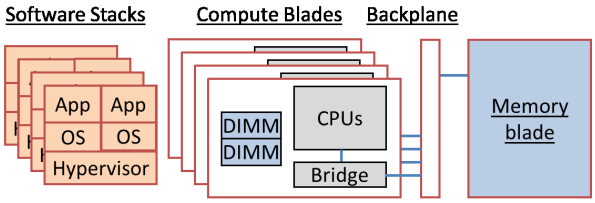

Figure 1: System diagram with memory blades

Remote memory access can be supported either in hardware by cache-line granularity through the cache coherence protocol or in software by swapping with local pages through DMA transfers. In this paper, we assume the software-based page swapping design because it requires minimal hardware changes and, as we demonstrate, performs well. Here, access to data stored in remote memory results in the entire remote page being transferred to local memory, which can then be directly accessed, and a local page being selected for eviction to remote memory. Additionally, to maximize memory utilization, the remote memory capacity can be dynamically allocated among the connected servers. This capability allows servers to flexibly adjust their allocated remote memory capacities based on dynamic demands.

As shown in Figure 1, this disaggregated memory design consists of a blade enclosure housing multiple compute blades, which connect over the backplane to one or more memory blades through a (PCIe) bridge and use the standard I/O interface to access the memory blade. Specifically, memory pages read from and written to the remote memory blade are transferred over the shared high-speed PCIe interconnect in the blade server enclosure. This design improves cost-efficiency by reusing the pre-existing commodity infrastructure and amortizing the cost of memory blade across multiple compute servers.

The memory blade, shown in Figure 2, consists of a custom ASIC or lightweight processor, storage for address mapping tables, a PCIe bridge for connecting over the backplane, and multiple rows of DIMMs attached through buffer-on-board or similar fan-out techniques. The ASIC or lightweight processor responds to memory accesses as they arrive from the PCIe interconnect. Servers accessing remote memory send a request for a memory location within their own address space, and the memory blade uses its address mapping tables to trans-

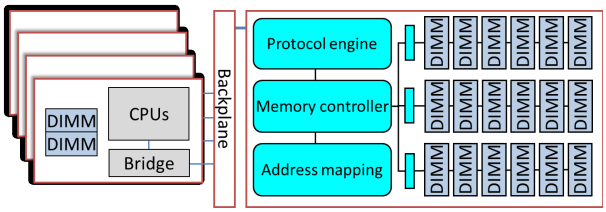

Figure 2: Memory blade architecture

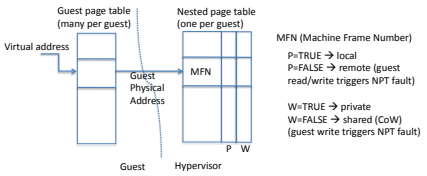

Figure 3: Page Tables

late the requested address and requesting server ID to the address of the data on the memory blade. In addition, either the processor or an external management processor runs management software that coordinates the dynamic capacity allocation, as well as discovery and setup phases for initializing the remote memory allocations as servers are booted up.

3 Software Architecture

To reach broad acceptance in the commodity market, disaggregated memory should provide benefits to legacy OSs and applications with minimal code modification. Our design extends a hypervisor to support remote memory, enabling arbitrary OSs and applications running in virtual machines to take advantage of the expanded capacity of remote memory without any code changes. Our design relies on existing processor support for full hardware virtualization.

We built a prototype system that extends the Xen hypervisor to support remote memory as a guest-transparent demand-paging store. The prototype sets aside a portion of a server's local memory to act as an emulated remote memory blade with a configurable speed. A future real memory blade could be substituted for this emulation mode by adding a simple device driver to the hypervisor.

Our extensions leverage support in modern processors for a new level of page table address translation called Nested Page Tables (NPT) by AMD and Extended Page Tables (EPT) by Intel (for simplicity, we generically use the term NPT in the following). The idea of adding this new level of translation is to enable fully virtualized guest VMs to maintain and update traditional OS page tables without hypervisor involvement. Guest page tables map guest virtual addresses to pseudo-physical addresses. These PFNs (pseudo-physical frame numbers) are in turn translated by the hardware to host machine physical addresses, called MFNs (machine frame numbers) using the NPTs, which are managed exclusively by the hypervisor.

We modified NPT handling to dynamically change the mappings of guest PFN to host MFN as memory pages are migrated back and forth on-demand between the servers local memory and a remotely accessed memory blade. The page table structure is shown in Figure 3. The hypervisor maintains one NPT for each guest VM. Any PFN with a corresponding machine page that has been migrated to the remote memory is marked not present in the NPT. Thus, a guest memory access to a remote page causes an NPT page fault in the hypervisor (not in the guest). The hypervisor retrieves the referenced page from the memory blade, marks the NPT entry present, and then resumes the faulting guest virtual CPU (vCPU). Unlike with paging to disk, access times to remote memory are fast enough that it is not worthwhile to de-schedule a faulting guest VM.

If a free local page is not available when a remote page needs to be migrated, a local page is selected for eviction and transfer back to remote memory. We experimented with well-known page replacement policies: *Round-Robin* and *Clock* (which approximates Least-Recently-Used by examining the NPT page table Accessed bit). Both are global replacement policies that consider all of local memory for eviction, as opposed to a per-VM replacment policy that would limit the choice to pages of the VM that caused a page fault. While per-VM replacement would limit cross-VM performance interference, a global replacement policy is more likely to uncover "cold" pages that will not be accessed in the near future.

A complementary technique for increasing effective memory capacity is content-based page sharing (CBPS) [5, 2, 4]. With CBPS, the hypervisor detects memory pages with identical content, both within a VM and across VMs, and transparently replaces the page copies by a single copy. Transparent sharing of identical pages increases the total effective memory capacity of the system. In the context of consolidation, CBPS helps free up memory to allow more VMs to be placed on a single machine. The opportunity for sharing is greatest when the VMs are similar, for example running the same OS kernel or having similar data sets.

To implement CBPS, we borrowed code from Difference Engine [2] to extend the Xen hypervisor to detect when guest memory pages have identical content and transparently create NPT mappings to a single shared MFN with read-only access (writeable bit set to false

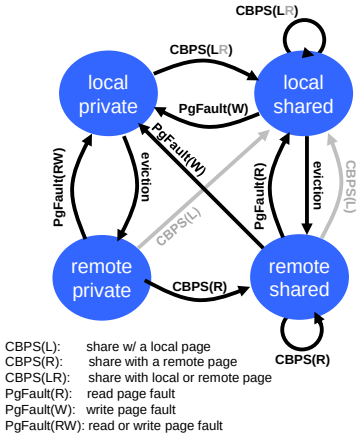

Figure 4: PFN State Transitions

in the NPT entries). The MFNs containing redundant copies are freed. (Unlike Difference Engine, we only share pages that are fully identical, and we do not compress page data). A guest write access to a shared MFN causes an NPT page fault in the hypervisor (again, not in the guest). The hypervisor creates a private writeable copy of the MFN and changes the guest PFN to MFN mapping for the faulting PFN. Then the hypervisor resumes the faulting guest vCPU.

Systems with CBPS require memory overcommit support, where the total physical address space seen by VMs exceeds the host machine's physical memory capacity. When page sharing is broken due to memory writes, new host memory pages are allocated and can overflow the host memory capacity. Typical CBPS systems would overflow to disk storage, but we use a memory blade for this purpose. The memory blade has data from multiple systems, offering significant opportunities for cross-host content sharing on the memory blade itself. Additionally, CBPS can be used at both the local and remote memory levels to increase overall effective capacity at both levels. By reducing the total active working set, sharing at the local level benefits remote memory by decreasing the amount of remote memory required.

The state transitions for a single guest PFN is shown in Figure 4, where greyed out transitions correspond to sharing opportunities between local and remote MFNs, an optional feature. Some state transitions in Figure 4 require MFNs to be allocated or freed. For example, when a write access to a shared page occurs, a new MFN needs to be allocated for a local private copy. To obtain this page, the hypervisor may first need to evict some local page by migrating its contents to remote memory. Thus, a PgFault(W) transition from local/remote shared to local private may trigger an eviction action that migrates another PFN/MFN from local private or local shared to

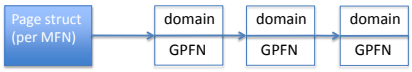

Figure 5: Inverse Mapping

remote private/shared.

When a shared page is moved to local or remote memory, the NPT entries for all PFNs that share the page need to be updated to ensure proper address translation and page fault triggering. We added a reverse map data structure (Figure 5) that records the PFNs sharing each MFN. Entries are added when CBPS page scanning detects a sharing opportunity, and removed on guest write accesses causing page faults. These actions can occur concurrently by multiple CPU cores. To avoid lock contention, which we found to be expensive for this structure, we designed it to be lock free, allowing concurrency of multiple delete operations and one add operation. Deletes mark entries in the inverse map as invalid rather than de-linking the entries, and entries are later garbage collected by the single CPU core that adds entries when performing periodic CBPS scans. Additional concurrency control challenges had to be overcome; for example, a page fault caused by one guest can trigger eviction that causes an NPT entry of another guest to be modified. Finally, pages used as source or destination of DMA transfers by I/O devices must be kept local and non-shared. To enforce these properties, we added code to Xen's QEMU-based I/O emulation (in a helper "stub domain") to intercept memory pages used by guest VM I/O operations and then prevent their eviction or sharing.

4 Evaluation

We ran our prototype hypervisor on a large memory system which has 8 quad-core AMD Opteron processors, with 256 GB of total RAM. We ran two primary workloads: VoltDB, an in-memory database software using a TPC-C-esque setup; and mstress, a microbenchmark we wrote which allots a virtual memory region for frequent access (hot) and a separate region for infrequent access (cold). VoltDB is setup with the database server VMs running on our test system, and another server running the client drivers which execute transactions against the database. Due to its network interactions and in-memory design, network and memory performance are highly stressed. Mstress is highly memory access-intensive and designed to stress our system. We ran using 10^8 accesses to 7 GB of memory; the hot memory region is 0.7GB and the cold region is 6.3GB, with 90% of accesses going to hot pages. Half of the hot pages are shareable with two copies of each shared page.

We use the VoltDB workload to illustrate the perfor-

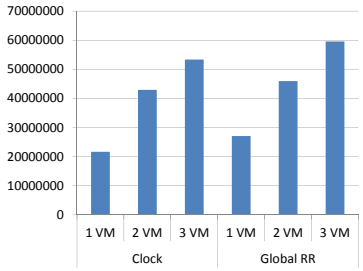

Figure 6: Remote accesses in mstress

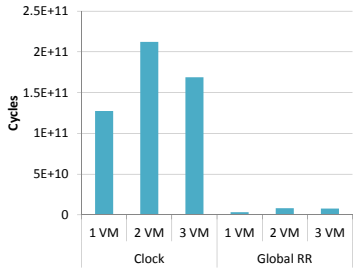

Figure 7: Time in mstress selecting victim pages

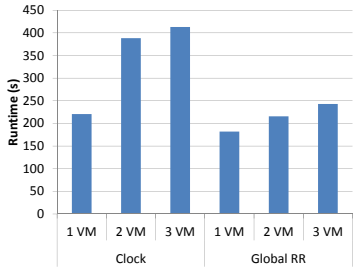

Figure 8: mstress runtimes

mance of disaggregated memory and CBPS on memory intensive workloads, while we use the mstress workload to provide insight into the fine-grained, function-level performance of our prototype. A single workload instance is run per-VM, and the VoltDB, and mstress VMs have 4 and 8 GB of memory, respectively. In addition to our prototype, we also developed a cost model to determine the benefits provided by disaggregated memory versus baseline systems. This model estimates the 3 year total cost of ownership of a system, factoring in the primary hardware components (CPU, memory, disk, memory blade, etc.), as well as the 3 year power cost.

4.1 Page Replacement Policy Impact

We used the mstress workload to study the impact of page replacement policy. We ran the workload using Clock and Round Robin with 1, 2, and 3 VMs. Figure 6 plots the total number of remote memory accesses incurred with each run. Overall, the number of remote accesses is lower with Clock than Round Robin, indicating that Clock is better able to select victim pages that are unlikely to be accessed soon. On the other hand, Figure 7 plots the total amount of CPU cycles spent in each run to select a victim page, and the resulting workload runtimes are shown in Figure 8. Executing the Clock algorithm is far more expensive than Round Robin. Although Clock makes slightly better decisions than round robin about which pages to evict, Clock must examine more pages and access more data structures to make each decision. Compared to disk-based paging, where it pays to spend a few extra CPU cycles to make more optimized choices, the much lower latency of memory blades changes that traditional trade-off in favor of making fast, less optimized decisions.

4.2 VM Consolidation Results

Using the VoltDB workload, we measured the performance, in terms of throughput, of several configurations. We ran between 1 to 8 VoltDB VMs, and considered five server configurations: Max, Base, CBPS, DM, and DM+CBPS. The Max configuration has enough RAM to host all of the VMs, Base is a cost-optimized server that has RAM sufficient for half of the VMs, and CBPS is the Base configuration with content based page sharing enabled to provide a greater effective memory capacity. The DM configuration uses our disaggregated memory solution with enough RAM to host all of the VMs, but half of the RAM is local to the server, and half is hosted on the memory blade. DM+CBPS uses the combination of disaggregated memory and content based page sharing.

The performance results are shown in Figure 9. We can see Max provides near linear performance scaling with VMs and Base provides the same scaling, but is only able to host half of the VMs. The CBPS configuration enables memory capacity savings of approximately 40%, allowing an additional virtual machine to be hosted. However, there is a slight performance penalty due to the time required to do page scanning and comparisons, as well as the copy-on-write breaking of shared pages if one of the sharers needs to modify the page. The DM configuration is able to provide performance within 10% of the Max configuration, demonstrating the effectiveness of our disaggregated memory design in keeping the working set in the local memory. At 8 VMs, DM shows a performance drop off, which is partially from the working set growing too large, and partially from some inefficiencies in our software that leads to serialization delays as more VMs are added. In future revisions, we expect these serialization delays can be significantly re-

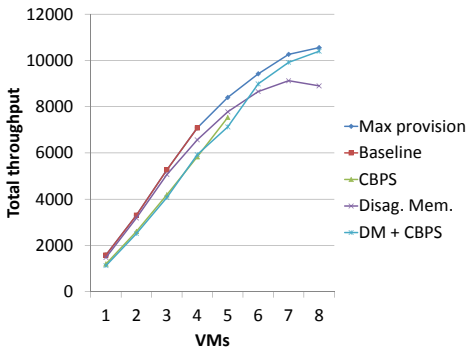

Figure 9: Total throughput as number of VMs is increased.

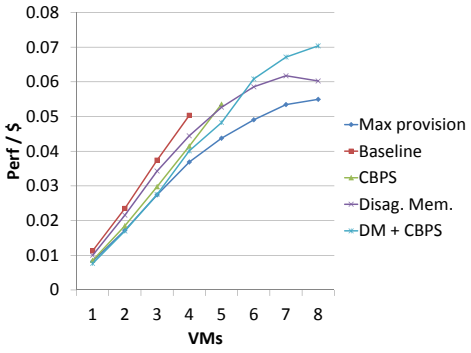

Figure 10: Performance-per-dollar results, factoring in the cost of each configuration

duced through more optimized data structures. Finally, the DM+CBPS provides slightly lower performance at 4 VMs compared to DM, due to the same performance penalties as in the CBPS case, but provides better scaling at higher VMs, as the 40% memory capacity savings frees up local memory to store a larger portion of the working set.

In Figure 10, we show the performance-per-cost results for each of the configurations. Base is able to provide the best perf/$ for 1-4 VMs due to being a cost-optimized server, but it is unable to scale to higher VMs and provide as high peak perf/$ as the other configurations. The CBPS option, due to its lower performance, provides lower perf/$ at the same number of VMs as Base, but is able to scale to a higher number of VMs. The DM and DM+CBPS options are able to provide a higher perf/$ compared to the Max configuration by utilizing less expensive DIMMs, as well as reducing processor and board costs by requiring less memory channels and less DIMMs per channel.

5 Related Work

Ye et al. [6] take a similar approach to extend the existing memory management indirection of an existing hypervisor to model an additional memory hierarchy level used transparently by OSs and applications in virtual machines. Our approach differs in that it takes advantage of recent processor hardware support for nested paging. More significantly, our investigation sheds light on the use case of server consolidation with memory disaggregation.

In addition, our work integrates memory disaggregation and content-based page sharing. CBPS has been the topic of multiple publications [5, 2, 4], while this paper demonstrates the complementary benefits of CBPS and emerging hardware techniques.

The recent MemX project [1] implemented a remote memory pager within a VM, with memory capacity provided by other servers in a cluster using Ethernet or Infiniband as the interconnect. While motivated by similar issues, our work differs in providing guest VM-transparent access to remote memory, while MemX exposes remote memory to guest VMs as a paging device. For guest VMs with MemX to take full advantage of the performance properties of remote memory compared to traditional storage media would likely require further modification of guest OS paging code. We also assume a different hardware platform that uses PCIe as the interconnect to a dedicated memory blade, resulting in a remote memory that is an order of magnitude lower latency than that provided by MemX, and therefore different software-level requirements. The most recent MemX work includes data deduplication (content based page sharing), but only performs the deduplication on any remotely stored pages. Our work examines the synergy between deduplication at both the local and remote memory levels.

6 Conclusions

Our work shows that disaggregated memory has strong potential benefits for server consolidation. It is able to provide high performance relative to a server using entirely local memory, and able to do so at a superior performance-per-cost. By combining disaggregated memory with content based page sharing, we have shown further benefits in freeing up local memory, enabling higher performance at higher VM consolidation than aggressive baseline systems.

References

[1] Deshpande, U., Wang, B., Haque, S., Hines, M., and Gopalan, K. Memx: Virtualization of cluster-wide memory. In

ICPP'10: Proceedings of the 39th International Conference on Parallel Processing (2010), pp. 663–672.

[2] GUPTA, D., LEE, S., VRABLE, M., SAVAGE, S., SNOEREN, A. C., VARGHESE, G., VOELKER, G. M., AND VAHDAT, A. Difference engine: harnessing memory redundancy in virtual machines. In *OSDI'08: Proceedings of the 8th USENIX conference on Operating systems design and implementation* (Berkeley, CA, USA, 2008), USENIX Association, pp. 309–322.

[3] LIM, K., CHANG, J., MUDGE, T., RANGANATHAN, P., REINHARDT, S. K., AND WENISCH, T. F. Disaggregated memory for expansion and sharing in blade servers. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture* (New York, NY, USA, 2009), ACM, pp. 267–278.

[4] MIŁÓS, G., MURRAY, D. G., HAND, S., AND FETTERMAN, M. A. Satori: enlightened page sharing. In *USENIX'09: Proceedings of the 2009 conference on USENIX Annual technical conference* (Berkeley, CA, USA, 2009), USENIX Association, pp. 1–1.

[5] WALDSPURGER, C. A. Memory resource management in vmware esx server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation* (New York, NY, USA, 2002), ACM, pp. 181–194.

[6] YE, D., PAVULURI, A., WALDSPURGER, C. A., TSANG, B., RYCHLIK, B., AND WOO, S. Prototyping a hybrid main memory using a virtual machine monitor. In *ICCD* (2008), IEEE, pp. 272–279.