Disaggregated Memory Architectures for Blade Servers

by

Kevin Te-Ming Lim

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2010

Doctoral Committee:

Professor Trevor N. Mudge, Co-Chair
Associate Professor Steven K. Reinhardt, Co-Chair
Professor Peter M. Chen
Professor James S. Freudenberg
Parthasarathy Ranganathan, Hewlett-Packard Labs

To my family and my Uncle Michael

Acknowledgements

The completion of this thesis and my Ph.D. bookmarks an important chapter in my life. I have had a lengthy stay at the University of Michigan, where I have proudly completed my bachelor's, master's, and (soon) Ph.D. degrees. I have truly enjoyed my time at U of M and in the state of Michigan; when I moved here 13 years ago, I never expected to fall in love with the state and have so many important life events happen there. But it truly has been a blessing, and I will always look back on this time with fondness.

I have been fortunate to have so many brilliant people help, influence, and guide me during my entire academic career. I wish to thank Mark Brehob, whose classes fostered my interests in computer architecture, Peter Chen, who helped me to become a better teacher and student, and Thomas Wenisch, who showed me how one must be endlessly dedicated to their work to be successful. I also wish to thank Renato Santos and Yoshio Turner for their help, expertise, and scholarly challenging of my ideas, which has helped me to continually improve my research. I especially would like to thank Jichuan Chang, who spent many hours with me discussing research ideas, planning new manuscripts, and working on paper submissions. My graduate school work has been highly collaborative, and I would not have accomplished even half of it without everybody's hard work.

Above all, the biggest influence on my academic life has been my incredible group of advisors, Steve Reinhardt, Trevor Mudge, and Partha Ranganathan. They have all helped shape my ideas and research, and more importantly given me the tools and foundation to

be successful. I hope to one day have a modicum of their wisdom and insight. I especially want to thank Partha, who really helped guide me in graduate school at a time when I had lost my direction. Without my advisors I would not be where I am today.

I also wish to thank all of the other students I have had the pleasure of working with: Nathan Binkert, Ali Saidi, Lisa Hsu, Ron Dreslinski, Korey Sewell, and Gabe Black. Their friendship and camaraderie have helped make every day of a very lengthy graduate school a little more bearable.

Two of the most important people in my life are my parents, Henry and Mamie. They have supported me all throughout my life, allowing me to seek my own path (although gently hinting that they wanted me to be a medical doctor), and fostering my love of computers. I am truly grateful for their love, support, and security they have provided me. I also wish to thank my brother, Chris, and his wife, Catherine, and their adorable son, Julian. My brother has been a role model in my life, and I hope to live up to the examples set by my entire family.

My acknowledgements would not be complete without thanking the most important person in my life, my wife Natela. She is my closest friend and greatest supporter, and has helped me to become a better person. Having her in my life has changed my world in unexpected ways, all for the better. I know that the happiness I have in my life, and my completion of this thesis, would not be possible without her.

I wish to thank HP Labs for its generosity and the amazing opportunities it has given me. The generous funding from HP Labs has supported several years of my graduate school and research. The managers and directors at HP Labs, including Norm Jouppi, Chandrakant Patel, John Sontag and Rich Friedrich, have all helped mentor me and built

a great work environment. I also wish to thank AMD for allowing me to intern there for a summer, and my mentor there, Ravi Bhargava, who introduced me to the rigors of industry and academia. Last, but definitely not least, I wish to thank the numerous friends and family in my life who have all supported me in my studies.

Table of Contents

List of Figures

List of Tables

Abstract

Disaggregated Memory Architectures for Blade Servers

by

Kevin Te-Ming Lim

Co-Chairs: Trevor N. Mudge and Steven K. Reinhardt

Current trends in memory capacity and power of servers indicate the need for memory system redesign. Memory capacity is projected to grow at a smaller rate relative to the growth in compute capacity, leading to a potential memory capacity wall in future systems. Furthermore, per-server memory demands are increasing due to large-memory applications, virtual machine consolidation, and bigger operating system footprints. The large amount of memory required is leading to memory power being a substantial and growing portion of server power budgets. As these capacity and power trends continue, a new memory architecture is needed that provides increased capacity and maximizes resource efficiency.

This thesis presents the design of a disaggregated memory architecture for blade servers that provides expanded memory capacity and dynamic capacity sharing across

multiple servers. Unlike traditional architectures that co-locate compute and memory resources, the proposed design disaggregates a portion of the servers' memory, which is then assembled in separate memory blades optimized for both capacity and power usage. The servers access memory blades through a redesigned memory hierarchy that is extended to include a remote level that augments local memory. Through the shared interconnect of blade enclosures, multiple compute blades can connect to a single memory blade and dynamically share its capacity. This sharing increases resource efficiency by taking advantage of the differing memory utilization patterns of the compute blades.

This thesis evaluates two system architectures that provide operating system-transparent access to the memory blade; one uses virtualization and a commodity-based interconnect, and the other uses minor hardware additions and a high-speed interconnect. The ability to extend and share memory can achieve orders of magnitude performance improvements in cases where applications run out of memory capacity, and similar improvements in performance-per-dollar in cases where systems are overprovisioned for peak memory usage. To complement the evaluation, a hypervisor-based prototype of one system architecture is developed. Finally, by extending the principles of disaggregation to both compute and memory resources, new server architectures are proposed for large-scale data centers that can double performance-per-dollar when considering total cost of ownership compared to traditional servers.

Chapter 1

Introduction

Current hardware and software trends indicate an impending memory capacity wall that is hindering the full utilization of per-server compute resources. Compute capacity is growing at a fast rate due to the increasing number of central processing unit (CPU) cores per chip. Designs from AMD and Intel will have as many as 8 CPU cores per chip in the near future, and these numbers are expected to continue to increase [9]. To effectively utilize this growing compute capacity, there needs to be a matching increase of memory capacity. Further increasing the pressure on capacity, memory requirements of applications and operating systems are rising, with examples including in-memory databases, search engine indices, OS footprint growth, and virtual machine consolidation [15]. These combined trends require increased scaling in memory capacity in order to meet hardware and software demands. However, aggregate projections of memory capacity growth per CPU socket show that capacity is unable to keep up with these demands [86]. Memory is already one of the most important resources in servers, and left unchecked, the impending memory capacity wall will lead to future systems being underutilized due to lack of memory capacity.

Concurrently, the memory system is becoming a larger contributor to total server power and costs [29] due to the high operating speeds of memory and large memory

capacities that must be provisioned for expected workload requirements. This problem is magnified by the many servers in large scale data centers. In recent years, data centers have grown in size and importance to the computer industry [11, 68]. One of the primary factors of their growth is the emergence of the internet sector, which has large-scale applications such as web search. These sectors have massive data centers, as evidenced by companies such as Google and Microsoft having hundreds of thousands of servers spread across several globally distributed locations [35]. At this scale, the power and cost efficiency of servers is closely linked to total operating costs. The large contribution of the memory system to both power and costs therefore makes it critical to have highly efficient and well utilized memory resources in servers across the entire data center.

This thesis examines the dual needs of addressing the memory capacity wall and having power and cost-efficient memory resources in large-scale environments. Given these needs, it is apparent that new memory architectures are required that can provide expanded memory capacity and more efficient memory resources. Due to their targeted use in large-scale environments, these architectures must be designed to exploit the economies of scale, taking advantage of the lower costs of components with high sales volume, while minimizing the use of custom components. At the same time, when designing these memory systems, there are new opportunities to significantly improve their effectiveness. One such opportunity is optimizing for the *ensemble*, which refers to a group of closely located servers. When viewed at an ensemble level, there is time-varying, differing memory usage across servers due to various applications being run and assorted workload inputs. This variability can be leveraged to provision resources for the expected sum of memory usage across the ensemble, as opposed to the current standard

practice of provisioning each server for the expected worst case memory usage. Additionally, the increased prevalence of blade servers—which have fast, shared interconnection networks—and virtualization software—with its support for hardware indirection—offers opportunities for efficient memory architectures that provide shared, expanded capacity in an operating system and application-transparent fashion.

In this thesis, these unique opportunities are explored in order to design a new, disaggregated memory architecture. The concept of disaggregation is used to refer to the separation of a resource from its traditional location within a compute server, and the assembly and use of that resource in a way that promotes utilization and cost efficiency across the ensemble. Specifically this thesis considers separating a portion of the memory resources from compute servers and organizing that memory in a cost and power-efficient manner that allows their utilization to be maximized across all of the servers. These memory resources are assembled in a specially designed *memory blade* that provides remote, expanded capacity to compute blades within an enclosure. The memory blade's capacity is dynamically allocated among the connected compute blades. The memory blade and two system architecture designs to access the blade are evaluated and are shown to effectively meet the goals of expanding capacity along with improving cost and power efficiency. A hypervisor-based prototype of one of the system architectures is implemented to provide greater understanding into the systems-level implications of disaggregated memory. Finally the principles of disaggregation are applied at a broader scale, and new server architectures are proposed for large-scale data centers that offer substantial performance and cost benefits over conventional servers.

Before discussing the disaggregated architectures in detail, a closer examination of the current memory capacity and power trends, new opportunities for memory system efficiency, and current work must be considered.

1.1 Memory Capacity Trends

In recent years, memory capacity has become a crucial yet limited resource in commodity systems. On the capacity *demand* side, current studies project an increased number of cores per socket, with some studies predicting a two-fold increase every two years [9]. As the number of cores per socket increases, the memory requirement to effectively utilize all of those cores also scales upward. Applications are requiring increasing amounts of memory capacity to deal with demands from web 2.0 applications, in-memory databases, and virtual machines [15]. Furthermore, operating systems are growing in memory footprint, with each successive generation of Windows requiring more memory [54]. However, from a *supply* point of view, memory capacity growth is unable to keep up. The International Technology Roadmap for Semiconductors (ITRS) estimates that the pin count at a socket level is likely to remain constant [86]. As a result, the number of channels per socket is expected to be near-constant. In addition, the rate of growth in dual in-line memory module (DIMM) density is starting to wane, growing at a rate of two-fold every three years versus the previous rate of two-fold every two years. Additionally the DIMM count per channel is declining (e.g., two DIMMs per channel on Double Data Rate 3 (DDR3) interfaces versus eight for DDR1 interfaces) [42].

Figure 1 aggregates these trends to show historical and extrapolated increases in processor computation and associated memory capacity. The processor line shows the projected trend of cores per socket, while the dynamic random access memory (DRAM)

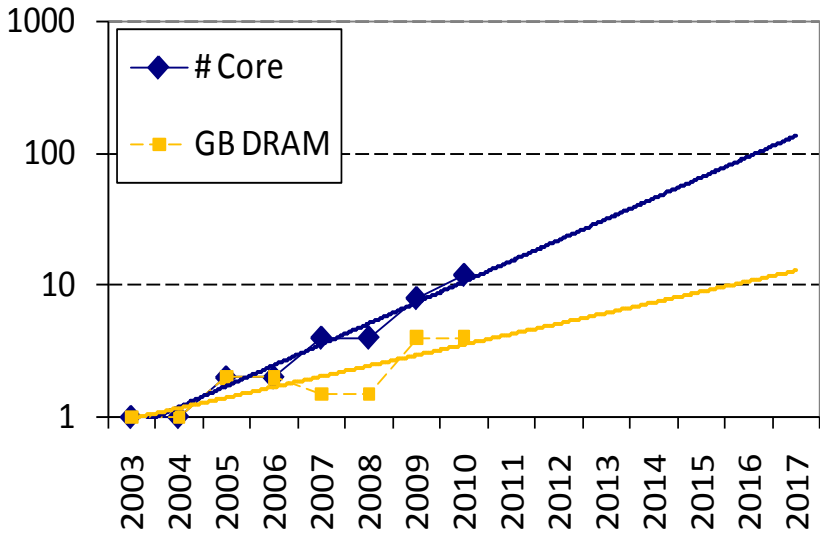
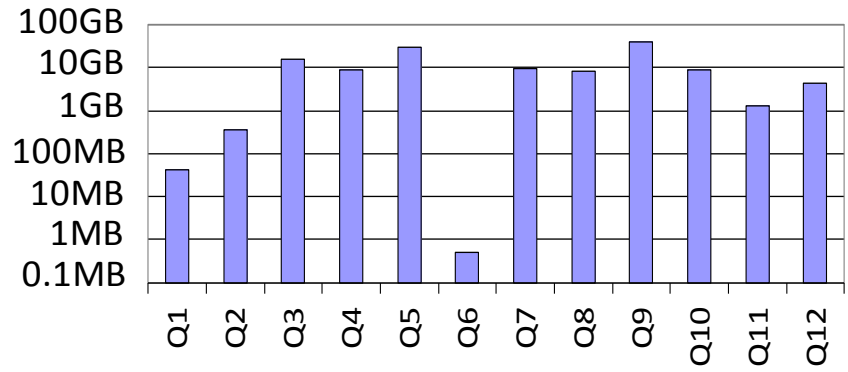


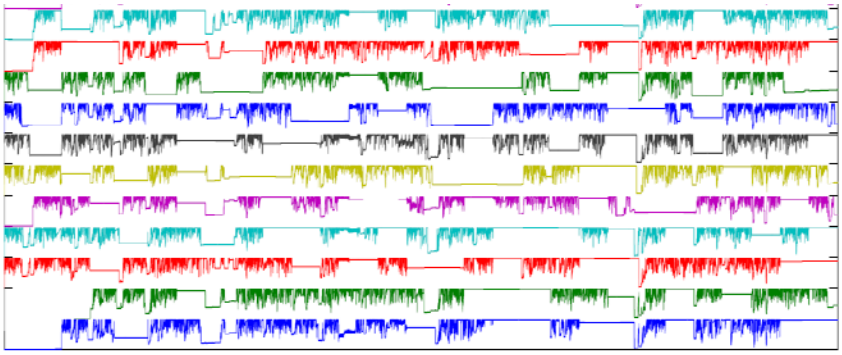
Figure 1: Projected annual growth in number of cores and memory capacity.
The expected number of cores per socket (blue line) is growing at a faster rate than the expected DRAM capacity (orange line). On average, memory capacity per processor core is extrapolated to decrease 30% every two years.

line shows the projected trend of capacity per socket, given DRAM density growth and DIMM per channel decline. If these trends continue, the growing imbalance between supply and demand may lead to commodity memory capacity per core dropping by 30% every two years. If not addressed, future systems are likely to be performance-limited by inadequate memory capacity.

At the same time, several studies show the contribution of memory to the total cost and power consumption of future systems increasing from its current value of about 25% [56, 67, 61]. A survey of memory costs of differing technology types and capacities, discussed in greater detail in Chapter 6, shows that the costs of higher speed, higher capacity memory can easily rival the cost of the processor. New generations of memory reduce power consumption, but these reductions are offset by faster operating frequencies and the larger memory capacities that are being used. Furthermore, certain recent



(a)



(b)

Figure 2: Intra- and inter-server variations in memory utilization.
(a) The amount of granted memory for TPC-H queries can vary by orders of magnitude. Note the memory usage is on a log sacle. (b) "Ensemble" memory usage trends over one month across 11 servers from a cluster used for animation rendering. Note the variability in utilization across the servers at any given time point.

memory technologies such as Fully-Buffered DIMMs (FB-DIMM) require additional hardware that increases power consumption [36]. As memory power trends higher, it will increase ownership costs as more energy will be required to adequately cool the servers.

There are new technologies emerging to improve memory capacity, including the use of alternative memory technologies such as Flash [94] or Phase Change Memory [57] as main memory, 3D stacked DRAM [51], and buffers [4] or ASICs [20] to increase the allowable DIMMs per memory channel. These technologies are all promising in increasing memory capacity, but each has different drawbacks, including asynchronous read/write performance, lifetime wear-out concerns, configuration flexibility, and non-

commodity costs. As a result, DRAM currently remains the primary option for main memory capacity despite these alternatives.

1.2 Heterogeneity Opportunities in Enterprise Workloads

These challenges in memory capacity and power present a unique opportunity for research and development. Recent studies have pointed to a new possibility to address these challenges—namely that of *optimizing for the ensemble* [75], referring to a group servers physically located near each other. For example, several studies have shown that there is significant temporal variation in the usage of resources like CPU time or power, both across and within applications. There are analogous trends in memory usage based on variations across application types, workload inputs, data characteristics, and traffic patterns. Figure 2(a) shows how the memory allocated for queries by a standard transaction processing benchmark, TPC-H, can vary widely. There are several orders of magnitude difference in memory allocated between the smallest and largest queries, demonstrating the large memory variations within an application. Figure 2(b) presents a high-level illustration of the time-varying memory usage of 11 randomly-chosen servers from a 1,000-CPU cluster used to render a recent animated feature film [1]. Each line illustrates a server's memory usage varying from a low baseline when idle to the peak memory usage of the application. As can be seen, there is variation in the memory usage across all servers at any time. These results point to the opportunity for a solution that provisions server memory for the typical usage, with the ability to dynamically add memory capacity across the ensemble. This solution can reduce costs and power by avoiding the over-provisioning that results from allocating each system for its worst-case memory usage.

1.3 Blade Servers

One solution for addressing resource efficiency for servers has been the blade server infrastructure. In these infrastructures, individual servers are in a “blade” form-factor, which is smaller than its 1U or 2U counterparts by excluding components such as fans, power supplies, and I/O devices. Instead, these blades plug into an enclosure which provides shared resources for those excluded components and other features such as network switches. By sharing resources, blade systems are able to be more efficient than traditional servers; for example, fans and power supplies can be sized for and operate in more efficient ranges. Because they are targeted towards enterprise environments, blade servers typically have support for management processors that allow remote control over server hardware and power settings. Additionally, blade vendors often include or support virtualization software to provide greater flexibility and control over the operating systems being run. By offering improved resource efficiency through sharing resources, blade servers offer unique possibilities for new memory system designs that address the previously discussed power and capacity concerns.

1.4 Thesis Statement

To address the memory capacity and power trends, this thesis explores new memory architectures that utilize the opportunities from ensemble-level optimizations and blade server infrastructures. Specifically this thesis presents the design and evaluation of a *disaggregated* memory architecture that is inspired by reassessing traditional memory systems which co-locate compute and memory resources. Instead, a portion of memory resources are separated from servers and are organized in a shared, remote memory pool,

and the servers' memory hierarchies are extended to include a remote memory level. This pool, provided through a capacity-optimized *memory blade*, is shared among compute servers within an enclosure to provide dynamic memory capacity allocation. Through memory blade-specific power optimizations, and by right-provisioning memory capacity at an ensemble level, disaggregated memory can address the power and cost problems that plague current servers. Compared to state of the art solutions, the memory blade is unique in enabling both capacity expansion as well as dynamic capacity sharing. By amortizing the memory blade costs across multiple servers in addition to its other benefits, the architecture offers significant cost advantages over current solutions that only expand capacity for a single server.

This thesis evaluates two system architectures that provide access to the memory blade and provide operating system-transparent capacity expansion. The first design requires changes only in hardware, while the second design limits changes to only the software layer. The disaggregated memory architectures are evaluated through simulations using a range of workloads, as well as utilization traces from multiple live-data centers, and are shown to provide on average a 10-fold performance improvement compared to a memory constrained baseline. To further elucidate the software requirements, a software-based prototype of disaggregated memory is developed by modifying the Xen hypervisor. The development leads to several interesting insights into the hardware-software interactions of disaggregated memory.

Finally, disaggregated memory is used as a cornerstone for more server architectures which uses compute disaggregation, optimized cooling and packaging, and low-power storage alternatives. These new servers are designed and evaluated for large-scale data

centers, specifically warehouse-computing environments, and are found to provide two-times higher performance per cost efficiency compared to traditional servers.

1.5 Contributions

This thesis makes the following contributions:

- Identification of memory capacity and power as system-limiting factors in future servers.
- Design of a *disaggregated* memory architecture for blade systems, providing an expanded and shared remote memory capacity.
- Evaluation using simulation of two implementations of disaggregated memory that focus on either hardware or software changes.
- Development and evaluation of a software-based prototype of disaggregated memory using the Xen hypervisor.
- Design of new, disaggregation-based server architectures that leverage low power components, redesigned packaging, and disaggregated memory.
- Evaluations demonstrating the effectiveness of the server architectures on novel workloads that model warehouse-computing environments.

1.6 Organization

The rest of the thesis is organized as follows: Chapter 2 contains background on many of the key concepts that are central to this work, including current memory architectures, servers, and blade systems. Chapter 3 discusses the disaggregated memory architecture, covering the design of the memory blade, and two system architectures that are used to access the memory blade. This architecture is evaluated primarily through simulation in

Chapter 4. The development of a software-based prototype of disaggregated memory is covered in Chapter 5. Chapter 6 explores the application of disaggregation to server architectures and evaluates these architectures on novel, data center-oriented workloads. Chapter 7 discusses related work in the memory and server architecture space. Chapter 8 presents future work and Chapter 9 concludes the thesis.

Chapter 2

Background

This chapter presents the background for several key topics, including main memory architectures, server designs and blade servers, and data centers. The motivation for this thesis is the limitation of current main memory architectures in providing adequate memory capacity and rising demands for that memory capacity. At the same time, inefficiencies in server architectures and the emergence of the blade infrastructure offer new opportunities for improving the resource efficiency of both memory architectures and servers. Finally, background on data centers and their growing importance to the computer industry is discussed. Based on growth of large-scale data centers, they will likely be a driver of server-class computer design in the near future.

2.1 Current Main Memory Architectures

2.1.1 Main Memory Architecture and DRAM

Commodity-class x86-based servers typically have one or two CPU sockets connected to a northbridge chipset, or memory controller hub. The northbridge is connected to several banks of dual in-line memory module (DIMMs) slots, and provides the communication between the CPU and the dynamic random access memory (DRAM) that is placed into the DIMM slots. In more recent processors, such as the AMD Opteron and

Property	DDR1	DDR2	DDR3
Maximum Transfer Speed	400 Mb/s	1066 Mb/s	1600 Mb/s
Voltage	2.6V	1.8V	1.5V
Maximum Density	1 Gb	4 Gb	4 Gb

Table 1: Memory technology properties of Double Data Rate (DDR) memories.

Phenom, and the Intel Core i7 and latest Xeon, the northbridge is integrated onto the processor die, thereby directly connecting the processor to the DIMM slots. The on-die northbridge configuration provides lower latency, faster access to the DRAM, but is conceptually very similar to the traditional design with a discrete northbridge. In these commodity x86 servers, the main memory consists solely of DRAM, which comes in several interfaces including double dynamic rate (DDR), DDR2, DDR3, and Fully-Buffered DIMM (FB-DIMM). As shown in Table 1 [64], each successive DDR generation offers higher speed and all DDR types assume a parallel bus interconnect, while FB-DIMM assumes a serial point-to-point link. In addition, the successive DDR generations have supported increased DRAM densities and lower operating voltages, thereby reducing the power generated.

DDR interfaces use a parallel bus architecture that requires the processor's memory controller to be connected to all of the memory slots. Because of the electrical complexity involved in the parallel bus wiring, the bus architecture is a limiting factor in the total number of DRAM devices that can be connected, the density of those memories, and the transfer speed of the interconnect. Achieving higher memory capacity and faster speeds thus leads to expensive and complex electrical signaling. This electrical complexity is

one factor why server-class motherboards – which typically offer a large memory capacity – cost a significant amount higher than traditional desktop-class motherboards.

2.1.2 Recent Designs for Increased Capacity

The development of FB-DIMM was driven largely by the limitations of the traditional parallel bus memory architecture. FB-DIMM instead uses a serial, point-to-point link, requiring far fewer wires that run at a higher frequency. The high-speed serial interface allows more DIMMs to be connected to the processor and higher overall bandwidths. FB-DIMM modules use DDR2-based DRAM devices to provide memory capacity, but have an additional chip on each module called an Advanced Memory Buffer (AMB). The AMB is responsible for reading the data from the DDR2 memory devices and converting it to the serial interface. However, there are drawbacks to FB-DIMM primarily due to the AMB chip, which is located on every DIMM. The first drawback is increased memory access latency due to converting the parallel DDR memory interface to the FB-DIMM serial interface. Because there is a large performance difference in memory speed and processor speed, for many applications memory latency can be one of the main factors in overall performance. Thus the extra latency from FB-DIMM can potentially result in slower performance compared to architectures using DDR interfaces. The second drawback is that the AMB draws a large amount of power [36]. Especially in servers with large memory capacities, where memory power can be quite high and even comparable to overall processor power, the power required for the AMB exacerbates the situation. Higher power consumption affects cost in terms of both higher power costs to run the systems, as well as higher cooling costs for the extra heat that is generated. Finally, the AMB itself adds to the cost of each of the DIMMs, resulting in higher prices than DDR

DIMMs of comparable speed and capacity. The added costs reduce the usefulness of FB-DIMM for high-volume markets, such as large-scale data centers. Furthermore, the costs, high power, and increased latencies point to the inadequacies of FB-DIMM to fully address the memory capacity needs faced by current memory architectures.

Although FB-DIMM is unlikely to become a ubiquitous solution – for example, AMD does not include FB-DIMM on its technology roadmap [23] – it does highlight several key needs for future memory architectures. First, memory latency is an important component for overall performance, and new architectures should avoid increasing latency in the common or most frequent access paths. Second, new technologies are needed to expand memory capacity beyond the current limitations faced by the parallel bus architecture. As will be described in the next section, it is necessary to have more memory capacity to match the compute capacity that is increasing with the growth of multi-core processors. Finally, memory power is an important consideration, especially for server-class computers. Resources must be efficiently utilized to maximize the amount of work done per watt.

Beyond FB-DIMM, there have been several newly proposed memory technologies to attempt to increase memory capacity. AMD has announced a product called G3MX [4], which is similar to FB-DIMM, but instead places the buffer chip on the motherboard. This configuration avoids the cost and power overheads that are incurred by an AMB being on every FB-DIMM. However, this technology has not been introduced into the market as of 2010; it is unclear if there will be a high-volume market that will drive commoditization costs; and including the buffers on the motherboard will increase motherboard costs.

Cisco has released an Extended Memory Blade server [20] which utilizes an ASIC plugged into a separate CPU socket to provide expanded capacity. The ASIC allows multiple DIMMs to be mapped into a single DIMM slot, surpassing the normal electrical limitations of standard processor memory channels. However, this architecture imposes non-commodity costs on all servers that require large memory, requires custom hardware and motherboard from Cisco, and is unlikely to reach high-volume markets.

Another technology is 3D stacked memory [51], which incorporates DRAM chips directly onto the processor die. This stacking can likely provide greater densities than is available with current DRAM, but like the other solutions incurs significant non-commodity costs. Additionally, the 3D stacked memory architecture could be more limiting in the possible memory configurations available if the only processor can only access the 3D stacked DRAM as main memory. This architecture would require a new processor model in order to obtain greater memory capacity, which would discourage high-volume markets that may have wide variety of memory capacity needs.

Although each of these solutions offer potential memory capacity increases, most offer only a one-time increase. In addition, they suffer from not being applicable to high-volume markets, making them inappropriate for cost-effective designs. The memory blade, on the other hand, minimizes the non-commodity parts and amortizes their costs across multiple servers, enabling cost-effective capacity expansion. Additional benefits are gained through dynamically sharing the memory blade capacity. Importantly, most of these technologies are orthogonal to the memory blade design, and can be utilized by the memory blade if commoditization drives their costs down.

2.1.3 Distributed Shared Memory Architecture

A processor's memory controller is typically connected directly to the accessible DIMM slots. However, in a distributed shared memory (DSM) system, multiple processors are connected to each other using a coherent interconnect, and processors can address other processors' memory in a single global, unified address space. Server-class processors such as the AMD Opteron and Intel Xeon support DSM through a multi-socket architecture, in which a single motherboard has two or more processor sockets connected by a coherent interconnect. These configurations are known as symmetric multiprocessing (SMP), where several identical processors share main memory. Most systems are limited to 4 sockets [23], but with additional circuitry and logic, much larger systems can be created, such as those historically offered by SGI [55]. In a DSM system, when a processor must access another processor's memory, the original processor generates a memory request. The memory request is sent to the processor connected to the memory, which completes the request on behalf of the original processor and returns the data. Memory contents across the different memories, as well as the caches of the processors, are kept up-to-date using a coherency protocol.

Distributed shared memory systems can provide a larger memory capacity than is possible with a single processor socket by sharing the memory of several processors. However, there are several drawbacks to increasing capacity by using a DSM system. First, latency to access another processor's memory is higher than accessing local memory because the request must traverse sockets to the other processor, and must take part in the coherency protocol to get the proper data. The increased latency has motivated the development of multiple schemes that utilize either hardware or software to minimize

the latency. Second, multi-processor motherboards cost more than single-processor motherboards due to their extra materials, complex electrical wiring required, and lower-volume market. Thus DSM systems can be a costly way to obtain greater memory capacity. Lastly, although DSM systems increase the total memory capacity, the capacity-per-socket ratio, or compute-to-memory ratio, is not changed. Because the compute-to-memory ratio is unchanged, although DSM systems will alleviate situations where the performance is memory capacity-bound, they will not address situations where each of the processors requires more capacity. Such situations will continue to have under-utilized compute resources.

2.2 Software Needs for Memory Capacity

To understand the needs for large memory capacities, current software resource demands must be examined. Due to the large gap between processor performance and disk performance, applications rely on main memory capacity to store and retrieve data with high performance. The reliance on main memory is especially true with the rise of latency-sensitive applications specifically in the internet sector. There has been an emergence of data-rich, highly interactive “web 2.0” applications where low latency, on the millisecond timescale for user response, is extremely important to overall user experience and thus crucial to end metrics such as total number of visitors to a website, or total number of order transactions. To support these applications – which are as widely varied as web search, social networking, blogs, and real-time chat – there has been significant effort to ensure they maximize the use of main memory and minimize the accessing of data from disk. These performance optimizations are evidenced by efforts of Google and Yahoo! to specifically make their web search indexes fit into main memory,

and the wide-spread use of a main memory based caching layer, *memcached* [32], in infrastructures such as Facebook or LiveJournal. In particular, memcached works by having a distributed cache that utilizes the main memory of servers to cache objects, such as the results of a database query or a page rendering. The servers running memcached can be standard servers running various applications, or can be dedicated caching servers. In large-scale infrastructures, there may be hundreds of computers dedicated to being memcached servers to provide high performance, and more importantly, fast response times. As web 2.0 infrastructures continue to grow and scale out, they will be one of the largest driving factors behind the need for greater memory capacity.

Similarly, there is also a growth in memory capacity demand in the database segment. Database management systems (DBMS) are seeing an increase in in-memory database systems (IMDS), which store an entire database in main memory [48]. This configuration is contrary to traditional DBMS, which store databases on disk. Significantly higher performance can be achieved by having the entire database in main memory because memory is an order of magnitude faster to access than disk both in terms of latency and bandwidth. This performance improvement is beyond what is achievable by caching a traditional disk-based DBMS in memory. Caching can speed up accesses to read objects by storing them in memory, but writing objects still pays the performance overhead of accessing the disk. Furthermore, IMDS are optimized to operate within memory, and avoid some of the overheads that disk-based DBMS have in maintaining data on disk.

Another key driver of memory capacity needs is virtualization software. Virtualization is increasingly being used to provide consolidation of servers in data centers. Virtualization enables software implementations of full-machines, called virtual

machines (VM), to be used in place of individual servers. Virtualization provides each VM the appearance of being its own machine through the use of a virtual machine monitor (VMM), or hypervisor. The hypervisor is a small layer of software that abstracts the underlying system's physical hardware in a manner that enables the VMs to be agnostic to actual hardware. This abstraction allows VMs to run (in the ideal case) on any hardware that is supported by the hypervisor without needing reconfiguration. Moreover, it allows multiple VMs to share a single machine by multiplexing the hardware at the hypervisor level. Operating systems typically expect to have full and sole control of the server's hardware, and thus servers normally only run a single operating system. Through the support and indirection of the hypervisor, virtualization can surpass this limitation and allow multiple operating systems to run simultaneously; for example, one VM can be running Windows XP, another VM can be running Debian Linux, and yet another could be running Red Hat Linux, all at the same time.

This ability to run multiple operating systems is especially useful for servers in data centers. Often times such servers are utilized only lightly, but cannot be consolidated because they are running applications that require separate servers. This limitation may be either due to compatibility reasons (e.g., one application needs Windows Server, and another needs Linux), application requirements, or security reasons (e.g., isolation). Without virtualization, each instance of these applications would require their own server, which comes with the costs of purchasing, running, and managing that server. With virtual machines, each of those applications can be run in a separate VM, and those VMs can be consolidated onto a single server, assuming it has enough compute, memory, and storage resources. Virtual machine consolidation can greatly reduce the amount of

running servers required in a data center, especially when the servers are not consistently under heavy utilization. Furthermore, the appearance of VMs as full systems can help ease implementation of services. For example, in Amazon's Elastic Compute Cloud [84], VMs give the users "full" systems which can be used for individual purposes.

Unfortunately, when consolidating virtual machines, memory capacity can be a bottleneck preventing further consolidation, especially for virtual machines with low levels of processor utilization. Consolidated VMs require the server have memory capacity to hold the entirety of their memory; thereby the consolidation of multiple VMs onto a single server greatly increases the memory resource pressure on that server. Thus in these VM environments, it is very important to have a large memory capacity to support the high levels of consolidation.

As discussed in this section, there are multiple software trends driving the need for increased memory resources – specifically memory capacity. Applications and software such as web 2.0, in-memory databases, and virtualization all require large memory capacities to achieve high overall system performance. As applications with large main memory requirements continue to become more prevalent, it will be important that new memory architectures are able to supply them with adequate capacity and resources.

2.3 Servers and Blade Systems

Server infrastructures are undergoing a design evolution that seeks to increase their efficiency and reduce costs, while also providing maximum flexibility. Blade servers are a prime example of such an evolution. To understand the need behind the design evolution, some background information on commodity-class servers must be reviewed. Servers found in data centers typically come in form-factors of 1U or 2U, which refers to

their vertical height being 1.75” or 3.5”, respectively. These servers are rack mounted, in which they are connected to a standardized chassis that measures 19” wide. Typical racks are 42U tall, allowing twenty-one to forty-two servers per standard rack. There are more advanced designs that allow mounting of servers on both the front and back of the rack, thereby doubling the number of servers per rack. A 1U or 2U server contains all traditional computer components, including motherboard, processor, RAM, I/O devices such as Ethernet network interface cards (NICs), and typically one or more hard drives. Also included are one or more power supplies (if more are needed for redundancy) and cooling fans to pull cold air from outside of the server, pass the air over the components, and exhaust the air out the other side. 1U servers are traditionally single or dual-socket systems, and are often the mainstay of large-scale data centers for their combination of cost, size, compute power, and flexibility. 2U servers offer more capacity for components, such as quad-socket systems, or multiple disk drives.

Blade systems are intended to improve upon the design of servers by offering greater flexibility in form-factor and configuration, and reducing the redundant resources among groups of servers. Instead of requiring servers to be a 1U or 2U form-factor, individual blade servers, or compute blades, plug into a rack-mounted enclosure, allowing wider options for form-factor and arrangement. The flexibility allows blade servers to better match the configurations needed by applications while requiring less physical space than traditional 1U servers. For example, the HP c-Class blade system has an enclosure that has a 10U form factor and can support up to 16 servers per enclosure. Blade infrastructures that are focused on server density, such as the HP Blade PC, can have up to 20 blade servers per 3U.

Blade systems often provide modularity in the type of blades that can be used. The server, or compute blade, is the most common form, but there can also be storage blades that contain disks, and I/O blades that contain I/O adapter cards such as graphics cards or networking cards. The blade enclosure contains a backplane that has electrical connectors that the blades physically plug into. Additionally the backplane provides resources such as power and interconnect (e.g., Ethernet and PCI Express). Blade servers are an important market segment for major companies such as HP, Dell, IBM, and Sun. Because blades are targeted towards enterprises, they often include advanced manageability features such as on-board management processors and power and cooling control, as well as reliability features such as redundant power supplies, fans, and networking.

Beyond having a more flexible form-factor, one of the key improvements of blade systems over traditional servers is providing a shared infrastructure. Traditional rack-mounted servers are isolated and each one has its own individual components. For example, each server has a power supply, cooling fans, and networking hardware. However, this isolation and redundancy can be wasteful – the power supplies are often forced to run in less efficient ranges of operation if servers are lightly utilized; cooling fans are forced to be smaller, less efficient form-factors to fit in a 1U or 2U case and may be underutilized depending on the activity level of the servers. Blade systems address these inefficiencies by providing power, cooling, and integrated networking resources at the enclosure level. This resource sharing allows for cooling fans to be larger and more efficient and the power supplies to be run at more efficient operating ranges (i.e., high utilization) by taking advantage of the varying utilization profiles of the compute blades

[58]. Additionally, the cost of the components is amortized across all of the compute blades that populate the enclosure.

2.4 Data Centers

Companies use data centers to house and run their large collections of servers. These data centers are buildings that have dedicated capacity to house, power, cool, and manage the servers. There has been significant growth in the data center segment, driven by the internet sector and cloud computing. Internet companies such as Microsoft, Google, and Yahoo! have invested millions of dollars in purchasing land, building, and running data centers, and continue to expand their capacity [93]. Similarly, there has been an emergence of *cloud computing*, in which third-party companies manage servers and infrastructure, and sell compute time and capacity to end users. The actual details of the infrastructure and management can be abstracted away from the end users, allowing them to focus only on the actual workloads being run. Cloud computing benefits the end users by allowing them to use as much compute power as necessary; as the users' applications require more processing power, more compute resources can be utilized from the cloud on an on-demand basis. With the emergence of cloud computing companies such as Amazon have dedicated large amounts of compute resources and are continually expanding their data centers.

The large scale internet sector and cloud computing data centers all have unique requirements compared to smaller deployments, and house thousands of servers running a variety of core applications or services such as web servers, web search, e-mail, and job schedulers. Due to this massive scale, efficiency at all levels of operation is very important. For example, millions of dollars have been invested in the development of

power-efficient data centers that are optimized to maximize cooling effectiveness and minimize power losses. Some recent data centers are being built in colder climates to take advantage of ambient external air cooling; other data centers have been built near power sources such as dams to provide cheap electricity [62].

Because of the enormous scale of these data centers, the design of domain-specific architectures can be feasible and cost-effective. Both Google and Yahoo! have their own optimized server enclosures [85]. Furthermore, companies have put millions of dollars into designing unique cooling infrastructures. There have also been efforts by Microsoft and Google in using shipping containers to house servers, providing a flexible and cost-effective building unit for data centers [35]. Based on the millions of dollars being put forth towards these efforts, and the sheer scale of these data centers, domain-specific architectures that offer significant power or cost advantages can be economically feasible.

Chapter 3

Disaggregated Memory Architecture

In order to address the memory capacity and power problems discussed previously, new architectural solutions are needed that can provide both transparent memory capacity expansion to match computational scaling, and transparent memory sharing at the ensemble level. In addition, given the large-scale data centers that these architectural solutions are targeted towards, it is important for these approaches to require at most minor changes to ensure that their costs do not outweigh their benefits. When considering the design of new memory architectures, blades servers offer an interesting platform due to their fast, shared interconnection networks and support of virtualization software. The increased market adoption of blade servers helps make them a viable platform for new memory architectures.

This thesis proposes a new architectural building block, a *memory blade*, which provides transparent memory expansion and sharing for commodity-based designs. The design of the memory blade is discussed in this chapter, and the memory blade is used to propose two new *disaggregated memory* architectures. Both architectures augment the memory hierarchy with a remote memory level whose capacity is provided by the memory blade. The first solution requires no changes to existing system hardware, using support at the virtualization layer to provide page-level access to a memory blade across a

commodity interconnect. My second solution proposes minimal hardware support on every compute blade, but provides finer-grained access to a memory blade across a coherent network fabric for commodity software stacks.

3.1 Disaggregated Memory Concept

There are four key observations that motivate the development of disaggregated memory:

(1) The emergence of blade servers with fast shared communication fabrics in the enclosure enables separate blades to share resources across the ensemble.

(2) Virtualization provides a level of indirection that can enable OS- and application-transparent memory capacity changes on demand.

(3) Market trends towards commodity-based solutions require special-purpose support to be limited to the non-volume components of the solution.

(4) The footprints of enterprise workloads vary across applications and over time; but current approaches to memory system design fail to leverage these variations, resorting instead to worst-case provisioning.

Based on these observations, this thesis argues for a re-examination of conventional designs that co-locate memory DIMMs in conjunction with computation resources. These designs connect DIMMs through conventional memory interfaces and the processor controls them through on-chip memory controllers. Instead, this thesis proposes a *disaggregated multi-level design* which provisions a separate memory blade, connected at the I/O or communication bus. The concept of disaggregation refers to the separation of memory from its location within the compute blade, and the use of that memory in a way

that promotes resource and cost efficiency across the ensemble. In this design, a portion of the memory resources are separated from the compute blades within an enclosure, and this memory is organized on a memory blade to maximize the efficiency across the whole ensemble. Thus the memory blade comprises arrays of commodity memory modules assembled to maximize density and cost-effectiveness. Importantly, the memory blade provides an extra, remote memory capacity to the compute blades. The memory hierarchy is reorganized to support this remote level in an operating system-transparent fashion through either modifications at the hypervisor level, or minor hardware changes. Furthermore, the memory blade takes advantage of varying memory requirements by allocating capacity on-demand to individual compute blades, allowing for more efficient overall utilization of memory.

3.2 Memory Blade Architecture

3.2.1 Memory Blade Design

The design of the memory blade is illustrated in Figure 3(a). The memory blade consists of a protocol engine, an interface bridge, a custom memory-controller ASIC (or a light-weight CPU), and one or more channels of commodity DIMM modules. The protocol engine and interface bridge are used to interface with the blade enclosure's I/O backplane interconnect, providing connectivity to multiple compute blades. The custom memory-controller handles requests from client blades to read and write memory, using standard virtual memory techniques to translate the requested memory addresses to their actual locations on the memory blade. Additionally the controller manages capacity allocation and address mapping across the connected compute blades. The memory blade's channels of DIMM modules use either on-board repeater buffers or alternate fan-

out techniques to allow each memory channel to support a high capacity. Optional memory-side accelerators can be added for functions like compression and encryption. The memory blade can either be designed as another blade in the system – supporting the same form factor as the compute blades – or could alternatively be designed to fit into the I/O bays typically found in blade enclosures. The former design would provide more physical space for DIMMs, while the latter design would avoid replacing any compute blades. The designs presented in this thesis are based on the latter design, where the memory blade is situated in the I/O bay to maximize the amount of compute capacity per enclosure.

The memory blade requires some custom hardware, including the memory-controller ASIC, the memory channels, and the motherboard. Although this custom hardware is contrary to the goal of avoiding non-commodity components, the memory blade requires no changes to volume blade-server designs, as it connects through standard I/O interfaces. In this thesis PCI Express® (PCIe®) or HyperTransport™ (HT™) are used as the I/O interconnect. Additionally, because the memory blade's capacity is available to all servers in the enclosure, its total costs are amortized over the entire server ensemble. Furthermore, the memory blade design is straightforward compared to a typical server blade, as it does not have the cooling challenges of a high-performance CPU and does not require local disk, Ethernet capability, or other elements (e.g., management processor, SuperIO, etc.). The simple design should minimize the cost of the custom hardware. To further offset hardware costs, the memory blade's DRAM subsystem is optimized for power and capacity efficiency. Because client access latency is dominated by the enclosure interconnect, resource efficiency optimizations are possible at the minor cost of

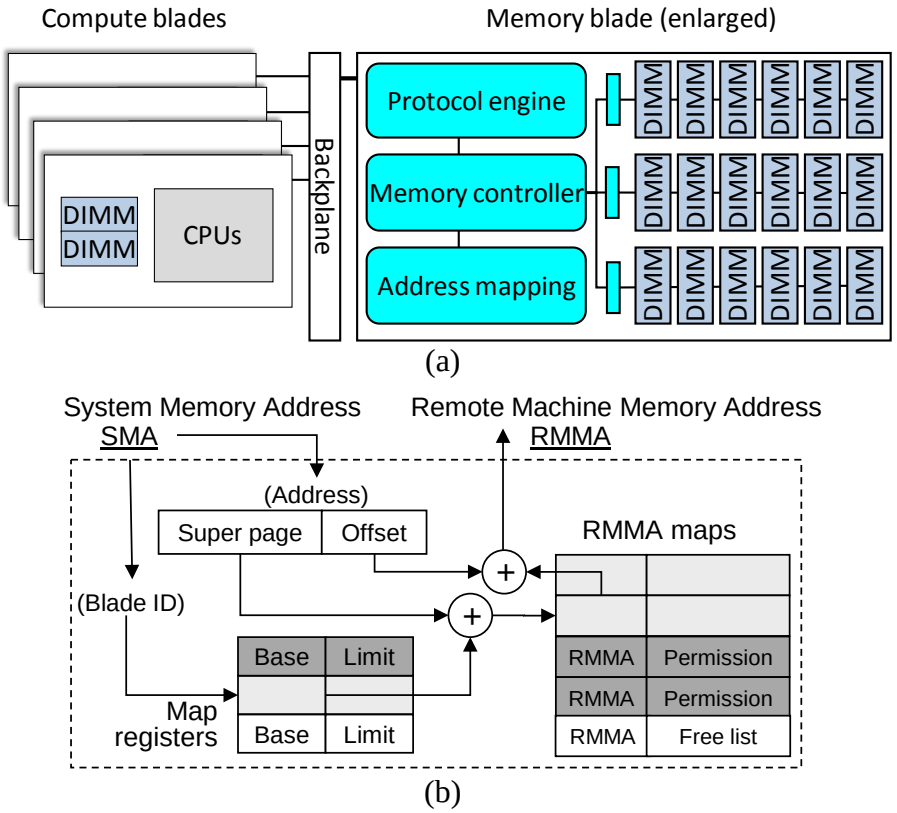


Figure 3: Design of the memory blade.

(a) The memory blade architecture is shown in detail. The memory blade connects to the compute blades via the enclosure backplane. (b) The address mapping data structures that support memory access and allocation/revocation operations.

performance. For example, the controller can aggressively place DRAM pages into active power-down mode, which reduces memory power by almost 90%, but takes an extra 6 DRAM cycles to transition into standby mode [63]. The controller can also map consecutive cache blocks into a single memory bank to minimize the number of active devices at the expense of reduced single-client bandwidth.

The design of the memory blade reflects deliberate choices to make a straightforward design that would fit well in the commodity-based markets, but is by no means the only feasible way a memory blade could be designed. The memory blade could possibly have greater compute power than an ASIC, making it closer to a large-memory server. This

design was not pursued in favor of simplicity as it would require greater complexity in the programming of the memory blade and its interactions with the compute blades. Although the memory blade uses either PCIe or HT, nothing precludes it from using other interconnects such as Infiniband® or QuickPath Interconnect. A memory blade could also serve as a vehicle for integrating alternative memory technologies, such as Flash or phase-change memory, possibly in a heterogeneous combination with DRAM, without requiring modification to the compute blades. This configuration would need more logic at the memory controller to communicate with the different memory types, as well as policies for using one memory technology versus another, but the added design complexity may be outweighed by the benefits from alternative memory technologies. Many of these alternative designs are discussed in Chapter 8.

3.2.2 Memory Blade Mapping

The memory blade is designed to be shared by multiple compute blades. To provide protection and isolation among shared clients, the memory controller translates each memory address accessed by a client blade into an address local to the memory blade, called the Remote Machine Memory Address (RMMA). In the proposed design, each client manages both local and remote physical memory within a single System Memory Address (SMA) space. Local physical memory resides at the bottom of this space, with remote memory mapped at higher addresses. For example, if a blade has 2 GB of local DRAM and has been assigned 6 GB of remote capacity, its total SMA space extends from 0 to 8 GB. Each blade's remote SMA space is mapped to a disjoint portion of the RMMA space. This process is illustrated in Figure 3(b). The blade's memory is managed in large chunks (e.g., 16 MB) so that the entire mapping table can be kept in SRAM on

the memory blade's controller. For example, a 512 GB memory blade managed in 16 MB chunks requires only a 32,000 entry mapping table. Using these "superpage" mappings avoids complex, high-latency DRAM page table data structures and custom translation lookaside buffer (TLB) hardware. This architecture does not preclude shared-memory communication among client blades, but in favor of a simplistic solution, it is not included in this initial design. The implications of supporting distributed shared memory are discussed in Chapter 8.

3.2.3 Remote Memory Allocation and Revocation

The memory blade's total capacity is partitioned among the connected clients through the cooperation of the virtual machine monitors (VMMs) running on the client servers, in conjunction with enclosure-, rack-, or datacenter-level management software. The VMMs in turn are responsible for allocating remote memory among the virtual machine(s) (VMs) running on each client system. The selection of capacity allocation policies, both among blades in an enclosure and among VMs on a blade, is a broad topic that deserves separate study. Here the discussion is restricted to designing the mechanisms for allocation and revocation.

Allocation is straightforward: privileged management software on the memory blade assigns one or more unused memory blade superpages to a client, and sets up a mapping from the chosen blade ID and SMA range to the appropriate RMMA range. Revocation is required when there are no unused superpages, and some existing mapping must be revoked so that memory can be reallocated. Capacity reallocation is a rare event compared to the frequency of accessing memory using reads and writes. Consequently, the design focuses primarily on correctness and transparency and not performance.

When a client is allocated memory on a fully subscribed memory blade, management software first decides which other clients must give up capacity, then notifies the VMMs on those clients of the amount of remote memory they must release. There are two possible approaches for freeing pages. First, most VMMs already provide paging support to allow a set of VMs to oversubscribe local memory. This paging mechanism can be invoked to evict local or remote pages. When a remote page is to be swapped out, it is first transferred temporarily to an empty local frame and then paged to disk. The remote page freed by this transfer is released for reassignment. Alternatively, many VMMs provide a "balloon driver" [95] within the guest OS to allocate and pin memory pages, which are then returned to the VMM. The balloon driver increases memory pressure within the guest OS by requesting memory, forcing the OS to select pages for eviction to satisfy the driver's request. This approach generally provides better results than the VMM's paging mechanisms, as the guest OS can make a more informed decision about which pages to page out to disk and may simply discard clean pages without writing them to disk. Because the newly freed physical pages can be dispersed across both the local and remote SMA ranges, the VMM may need to relocate pages within the SMA space to free a contiguous 16 MB remote superpage.

Once the VMMs have released their remote pages, the memory blade mapping tables may be updated to reflect the new allocation. It is assumed that the VMMs can generally be trusted to release memory on request; the unlikely failure of a VMM to release memory promptly indicates a serious error and can be resolved by rebooting the client blade.

3.3 System Architecture with Memory Blades

While the memory-blade design enables several alternative disaggregated memory architectures, two specific designs are discussed here, one based on making changes only at the software stack, and another based on requiring only minor hardware changes. Based on their implementation differences, the first design uses page swapping on remote access, while the second design provides fine-grained remote access. In addition to providing more detailed examples of possible disaggregated memory architectures, these designs also illustrate some of the tradeoffs in the multi-dimensional design space for memory blades. Most importantly, they compare the method and granularity of access to the remote blade (page-based versus block-based) and the interconnect fabric used for communication (PCI Express versus HyperTransport).

3.3.1 Page-Swapping Remote Memory (PS)

The first design avoids any hardware changes to the high-volume compute blades or enclosure; the memory blade itself is the only non-standard component. This constraint implies a conventional I/O backplane interconnect, typically PCIe. This basic design is illustrated in Figure 4(a).

Because CPUs in a conventional system cannot access cacheable memory across a PCIe connection, the system must bring locations into the client blade's local physical memory before they can be accessed. The Page-Swapping (PS) design leverages standard virtual memory mechanisms to detect accesses to remote memory and relocate the targeted locations to local memory on a page granularity. In addition to enabling the use of virtual memory support, page-based transfers exploit locality in the client's access stream and amortize the overhead of PCIe memory transfers. To avoid modifications to

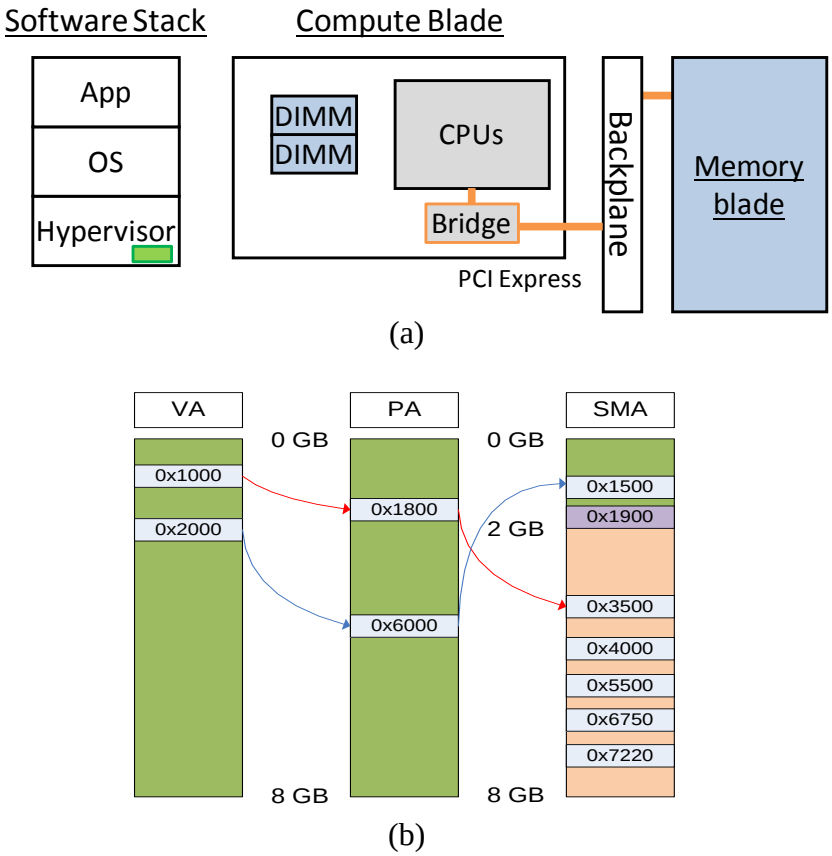


Figure 4: Page-Swapping (PS) Remote Memory Design.

(a) Disaggregated architecture that connects the compute and memory blades using a commodity interconnect. No changes are required to compute servers and networking on existing blade designs. Minor modules (shaded block) are added to the virtualization layer. (b) The address mapping design places the extended capacity at the top of the address space. In this example 2-8GB are remote memory.

application and OS software, this page management is implemented in the hypervisor or VMM. The VMM detects accesses to remote data pages and swaps those data pages to local memory before allowing a load or store to proceed.

The new page management scheme is illustrated in Figure 4(b). As mentioned previously, when remote memory capacity is assigned to a specific blade, the SMA (machine physical address) space is extended at that blade to provide local addresses for the additional memory. The VMM assigns pages from this additional address space to

guest VMs, where they will in turn be assigned to the guest OS or to the applications. However, because these pages cannot be accessed directly by the CPU, the VMM cannot set up valid page-table entries for these addresses. It instead tracks the pages by using "poisoned" page table entries without their valid bits set or by tracking the mappings outside of the page tables (similar techniques have been used to prototype hybrid memory in VMWare [98]). In either case, a direct CPU access to remote memory will cause a page fault and trap into the VMM. On such a trap, the VMM initiates a page swap operation. This simple OS-transparent memory-to-memory page swap should not be confused with OS-based virtual memory swapping (paging to swap space), which is orders of magnitude slower and involves an entirely different set of sophisticated data structures and algorithms.

The PS design assumes page swapping is performed on a 4 KB granularity, a common page size used by operating systems. Page swaps logically appear to the VMM as a swap from high SMA addresses (beyond the end of local memory) to low addresses (within local memory). To decouple the swap of a remote page to local memory and eviction of a local page to remote memory, a pool of free local pages is maintained for incoming swaps. The software fault handler thus allocates a page from the local free list and initiates a DMA transfer over the PCIe channel from the remote memory blade. The transfer is performed synchronously (i.e., the execution thread is stalled during the transfer, but other threads may execute). Once the transfer is complete, the fault handler updates the page table entry to point to the new, local SMA address and puts the prior remote SMA address into a pool of remote addresses that are currently unused.

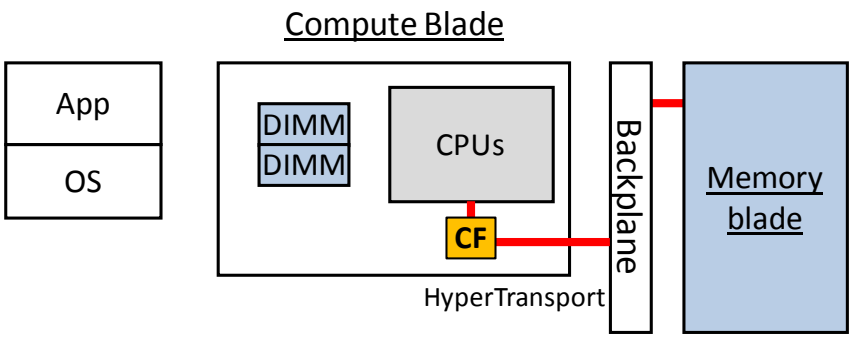


Figure 5: Fine-Grained Remote Memory Access (FGRA) Design.
This design assumes minor coherence hardware support in every compute blade. The added coherence filter hardware, CF, is shown.

To maintain an adequate supply of free local pages, the VMM must occasionally evict local pages to remote memory, effectively performing the second half of the logical swap operation. The VMM selects a high SMA address from the remote page free list and initiates a DMA transfer from a local page to the remote memory blade. When complete, the local page is unmapped and placed on the local free list. Eviction operations are performed asynchronously, and do not stall the CPU unless a conflicting access to the in-flight page occurs during eviction.

3.3.2 Fine-Grained Remote Memory Access (FGRA)

The previous solution avoids any hardware changes to the commodity compute blade, but at the expense of trapping to the VMM and transferring full pages on every remote memory access. The second design examines the effect of a few minimal hardware changes to the high-volume compute blade to enable an alternate design that has higher performance potential. In particular, this design allows CPUs on the compute blade to access remote memory directly at cache-block granularity.

This approach leverages the glueless SMP support found in current processors. For example, AMD Opteron™ processors have up to three coherent HyperTransport™ links

coming out of the socket. The Fine-Grained Remote Memory Access (FGRA) design, shown in Figure 5, uses custom hardware on the compute blade to redirect cache fill requests to the remote memory blade. Although it does require custom hardware, the changes to enable the FGRA design are relatively straightforward adaptations of current coherent memory controller designs.

This additional hardware, labeled “Coherence filter” in Figure 5, serves two purposes. First, it selectively forwards only necessary coherence protocol requests to the remote memory blade. For example, the coherence filter can respond immediately to invalidation requests because the remote blade does not contain any caches. Only memory read and write requests require processing at the remote memory blade. In the terminology of glueless x86 multiprocessors, the filter ensures that the memory blade is a home agent but not a cache agent. Second, the filter can optionally translate coherence messages destined for the memory blade into an alternate format. For example, HyperTransport-protocol read and write requests can be translated into generic PCIe commands, allowing the use of commodity backplanes and decoupling the memory blade from specific cache-coherence protocols and processor technologies.

Because FGRA allows the remote SMA space to be accessed directly by CPUs, VMM support is not required; an unmodified OS can treat both local and remote addresses uniformly. However, a VMM or additional OS support is required to enable dynamic allocation or revocation of remote memory. Performance can also potentially be improved by migrating the most frequently accessed remote pages into local memory, swapping them with infrequently used local pages—a task that could be performed by a VMM or by extending the non-uniform memory access support available in many OSes.

Chapter 4

Evaluation of Disaggregated Memory

This chapter evaluates the proposed disaggregated memory architectures, specifically examining the ability of the remote memory blade to effectively provide expanded capacity and shared capacity. The two system architectures for accessing the memory blade, PS and FGRA, are evaluated through simulation, and it is found that they are both able to provide high performance, but there are interesting trade-offs and some counter-intuitive results. The results show that memory disaggregation can provide significant performance benefits (on average 10X) in memory-constrained environments. Additionally, the sharing enabled by these solutions can enable large improvements in performance-per-dollar (up to 87%) and greater levels of consolidation (3X) when optimizing memory provisioning across multiple servers. Based on the results, some alternative designs are explored, specifically for the FGRA solution.

4.1 Methodology

4.1.1 Simulation Methodology

In this work, the performance of the memory blade designs is measured primarily via memory trace-based simulation because it makes it practical to process the billions of main-memory references needed to exercise a multi-gigabyte memory system. The

simulations are accomplished through a newly developed disaggregated memory simulator that uses detailed traces of main-memory accesses to estimate the memory blade's performance. These traces have information such as the address being accessed and the cycle of access. The simulator processes the traces and upon seeing a unique page, places that page in the first memory hierarchy level with free space (either local memory, remote memory, or disk, in that order). On subsequent access to the page, the simulator first determines the location in the hierarchy of the memory being accessed, and then simulates how long it would take to access the address based on any actions that must be taken (e.g., a remote page being accessed must first be swapped to local memory under the PS design). The simulator has multiple configurable parameters, including: size, latency and bandwidth of local memory, remote memory and disk; latency and bandwidth of the interconnect; hypervisor trap time; and packet processing time. The final output of the simulator includes a variety of statistics, such as peak bandwidth used, and the total cycles taken to complete the memory trace, which is the main performance indicator. Other possible alternatives to trace-based simulation are discussed in Section 4.7.1.

Memory reference traces were collected from the COTSon simulator, a detailed full-system simulator, used and validated in prior studies [8]. COTSon was modified to record the physical address, CPU ID, timestamp and read/write status of all main-memory accesses. To make it feasible to run the workloads to completion, a lightweight CPU model was used for this simulation. (Each simulation still took between 1 to 2 weeks to complete.) The simulated system has four 2.2 GHz cores, with per-core dedicated 64KB L1 and 512 KB L2 caches, and a 2 MB L3 shared cache.

Memory blade parameters					
DRAM Latency	120 ns	Map table access	5 ns	Request packet processing	60 ns
DRAM Bandwidth	6.4 GB/s	Transfer page size	4KB	Response packet processing	60 ns

(a)

Workloads		Footprint size
SPEC CPU 2006	5 large memory benchmarks: *zeusmp*, *perl*, *gcc*, *bwaves*, and *mcf*, as well as a combination of four of them, *spec4p*.	Low (zeusmp, gcc, perl, bwaves), Medium (mcf), High (spec4p)
nutch4p	Nutch 0.9.1 search engine with Resin and Sun JDK 1.6.0, 5GB index hosted on tempfs.	Medium
tpchmix	TPC-H running on MySQL 5.0 with scaling factor of 1. 2 copies of query 17 mixed with query 1 and query 3 (representing balanced, scan and join heavy queries).	Medium
pgbench	TPC-B like benchmark running PostgreSQL 8.3 with pgbench and a scaling factor of 100.	High
Indexer	Nutch 0.9.1 indexer, Sun JDK 1.6.0 and HDFS hosted on one hard drive.	High
SPECjbb	4 copies of Specjbb 2005, each with 16 warehouses, using Sun JDK 1.6.0.	High

(b)

Real-world traces	
Animation	Resource utilization traces collected on 500+ animation rendering servers over a year, 1-second sample interval. The presented data is from traces of a group of 16 representative servers.
VM consolidation	VM consolidation traces of 16 servers based on enterprise and web2.0 workloads, maximum resource usage reported every 10-minute interval.
Web2.0	Resource utilization traced collected on 290 servers from a web2.0 company. Traces are from sar with 1-second sample interval for 16 representative servers.

(c)

Table 2: Simulation parameters and descriptions of workloads and traces.

(a) The key simulation parameters used for the memory blade. (b) List of the workloads used for trace-based simulation (c) Details of the real-world memory utilization traces used in this thesis.

4.1.2 Simulation Parameters

The common simulation parameters for the remote memory blade are listed in Table 2(a). For the baseline PS, the memory blade interconnect is based loosely on a PCIe 2.0 x2 channel and has a latency of 120 ns and bandwidth of 1 GB/s (each direction). For the baseline FGRA, a more aggressive channel is assumed, e.g., based on HyperTransport™

or a similar technology, with 60 ns latency and 4 GB/s bandwidth. Additionally, for PS, each access to remote memory results in a trap to the VMM, and VMM software must initiate the page transfer. Based on prior work [79], it is estimated that the total software overhead is 330 ns (roughly 1,000 cycles on a 3 GHz processor), including the trap itself, updating page tables, TLB shootdown, and generating the request message to the memory blade. All of the simulated systems are modeled with a hard drive with 8 ms access latency and 50 MB/s sustained bandwidth. Initial data placement is performed using a first-touch allocation policy; pages are placed first in local memory until it is exhausted. Any subsequent new pages are placed into local memory after a local page is evicted to remote memory (if present), or disk.

The simulation model was validated on a real machine to measure the impact of reducing the physical memory allocation in a conventional server. The real machine tested was an HP c-Class BL465c server with 2.2GHz AMD Opteron 2354 processors and 8 GB of DDR2-800 DRAM. To model a system with less DRAM capacity, the Linux kernel was forced to reduce physical memory capacity using a boot-time kernel parameter. The findings from reducing the physical memory capacity were in qualitative agreement with the results obtained through trace-based simulation.

4.1.3 Workloads

The workloads used to evaluate the disaggregated memory architecture designs include a mixture of Web 2.0-based benchmarks (*nutch, indexer*), traditional server benchmarks (*pgbench, TPC-H, SPECjbb®2005*), and traditional computational benchmarks (*SPEC® CPU2006 – zeusmp, gcc, perl, bwaves, mcf*). Additionally a multi-programmed workload, *spec4p,* was developed by combining the traces from *zeusmp,*

gcc, *perl*, and *mcf*. *Spec4p* offers insight into multiple workloads sharing a single server's link to the memory blade. Table 2(b) describes these workloads in more detail. The workloads are further broadly classified into three groups—low, medium, and high—based on their memory footprint sizes. The low group consists of benchmarks whose footprint is less than 1 GB, medium ranges from 1 GB to 1.75 GB, and high includes those with footprints between 1.75GB and 3GB.

In addition to these workloads, the evaluation also uses traces of memory usage that were previously collected from two real-world, large-scale data center environments (*Animation*, and *VM consolidation*). To augment these traces, the memory usage in the data center of a photograph hosting and printing website was also collected (*web2.0*). All three data center traces are described in Table 2(c). The traces were each gathered for over a month across a large number of servers and are used to guide the selection of workloads to mimic the time-varying memory requirements of applications seen in real-world environments.

To quantify the cost benefits of the new memory designs, a cost model was developed for the disaggregated memory solutions and the baseline servers. Because the new designs target the memory system, data specific to the memory system is presented. Price data was gathered from public and industry sources for as many components as possible. For components not readily available, such as the remote memory blade controller, a cost range is estimated. Power and cooling costs are also included, given a typical 3-year server lifespan. DRAM power calculators are used to evaluate the power consumption of DDR2 devices [63]. Estimates for the memory contributions towards power and cooling are calculated by first calculating total memory power, and then calculating the total

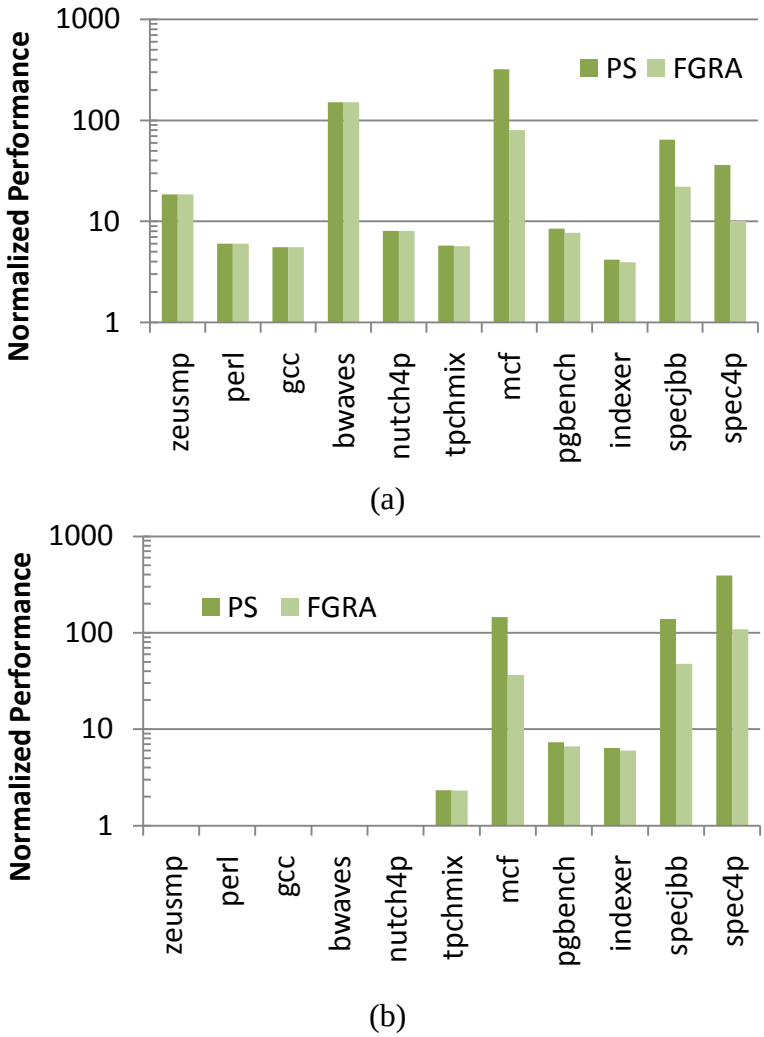


Figure 6: Capacity expansion results over memory-constrained baselines.
(a) Performance improvement of the two designs over `M-app-75%` provisioning; (b) Speedup over `M-median` provisioning

contributions of this memory power to the total power and cooling costs assuming a 3-year lifespan. The total power and cooling costs are calculated through previously published equations by Patel and Shah [71].

4.2 Memory Expansion for Individual Benchmarks

The first experiments focus on the applicability of memory disaggregation to address the memory capacity wall for individual benchmarks. To illustrate scenarios where

applications run into memory capacity limitations due to a core-to-memory ratio imbalance, each of the benchmarks are run on a baseline system with only 75% of that benchmark's memory footprint (M-app-75%). The baseline system must swap pages to disk to accommodate the full footprint of the workload. The baseline system results are compared with the two disaggregated memory architectures, PS and FGRA. In these cases, the compute nodes continue to have local DRAM capacity corresponding to only 75% of the benchmark's memory footprint, but have the ability to utilize capacity from a remote memory blade. The memory blade is allocated 32GB of memory as it is sufficient to fit any application's footprint. Figure 6(a) summarizes the speedup for the PS and FGRA designs relative to the baseline. Both of the new solutions achieve significant improvements, ranging from 4X to 320X higher performance. These improvements stem from the much lower latency of the remote memory solutions compared to OS-based disk paging. In particular, *zeusmp, bwaves, mcf, specjbb,* and *spec4p* show the highest benefits due to their large working sets.

Interestingly, PS outperforms FGRA in this experiment, despite the expectations for FGRA to achieve better performance due to its lower access latency. Further investigation reveals that the page swapping policy in PS, which transfers pages from remote memory to local memory upon access, accounts for its performance advantage. Under PS, although the initial access to a remote memory location incurs a high latency due to the VMM trap and the 4 KB page transfer over the slower PCIe interconnect, subsequent accesses to that address consequently incur only local-memory latencies. The FGRA design, though it has lower remote latencies compared to PS, continues to incur these latencies for every access to a frequently used remote location. Despite not supporting

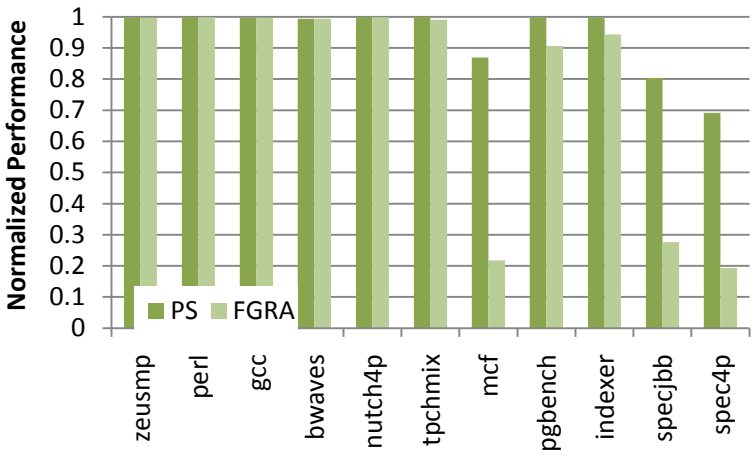


Figure 7: Capacity expansion results over worst-case provisioned baseline.
Slowdown of the two architectures versus worst-case (`M-Max`) provisioning.

page movement, `FGRA` outperforms the baseline. For further examination of the impact of locality, Section 4.6 considers an `FGRA` design that supports page swapping.

Figure 6(b) shows a possible alternate baseline where the compute server memory is set to approximate the median-case memory footprint requirements across the benchmarks (`M-median` = 1.5GB). This baseline models a realistic scenario where all servers are provisioned for the common-case workload, but can still see a mix of different workloads. Figure 6(b) shows that the proposed solutions achieve performance improvements only for benchmarks with high memory footprints. For other benchmarks, the remote memory blade is unused, and does not provide any benefit. More importantly, it does not cause any slowdown.

Finally, Figure 7 illustrates a baseline where the server memory is provisioned for the worst-case application footprint (`M-max` = 4GB). This baseline models many current datacenter scenarios where all servers are typically provisioned in anticipation of the worst-case load, either across workloads or across time. The memory disaggregation

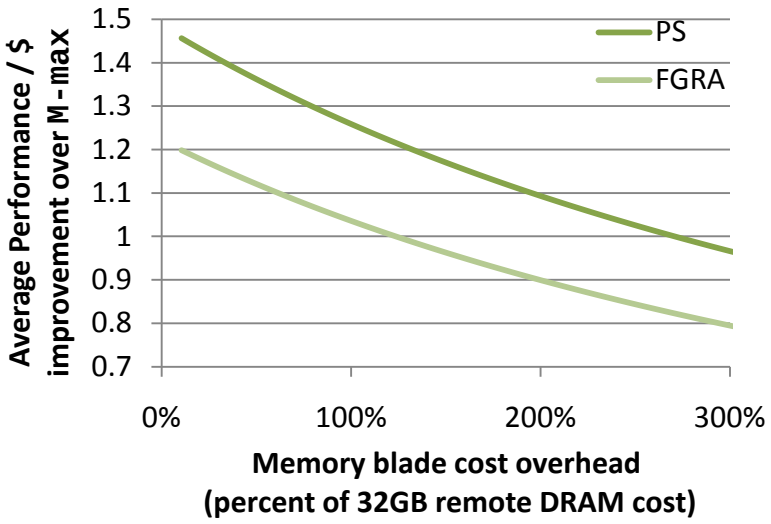


Figure 8: Cost analysis of the memory blade.
Average performance-per-memory dollar improvement versus memory blade costs relative to the total cost of 32GB of remote DRAM.

solutions are configured as in the previous experiment, with `M-median` provisioned per-blade and additional capacity provided by the remote blade. The results show that, for workloads with small footprints, the new solutions perform comparably. For workloads with larger footprints, going to remote memory causes a slowdown compared to local memory; however, `PS` provides comparable performance in some large-footprint workloads (*pgbench, indexer*), and on the remaining workloads its performance is still within 30% of `M-max`. As before, `FGRA` loses performance as it does not exploit locality patterns to ensure most accesses go to local memory.

4.3 Power and Cost Analysis

Using the methodology described in Section 4.1, the memory power draw of the baseline `M-median` system is estimated to be 10 W, and the `M-max` system is 21 W. To determine the power draw of the disaggregated memory solutions, it is assumed that local memory is provisioned for median capacity requirements (as in `M-median`) and there is

a memory blade with 32 GB shared by 16 servers. Furthermore, because the memory blade can tolerate increased DRAM access latency, the design aggressively employs DRAM low-power sleep modes. For a 16-server ensemble, the estimated amortized per-server memory power of the disaggregated solution (including all local and remote memory and the memory blade interface hardware, such as its controller, and I/O connections) is15 W.

Because the memory blade contains several custom designed components whose prices are not readily available, a range of costs is considered. Figure 8 shows the changes in the average performance-per-memory cost improvement over the baseline `M-max` system as the memory blade cost varies. The goal of this evaluation is to understand the memory blade's performance-per-memory cost benefit based on how much *cost overhead* the blade adds beyond the price of acquiring the remote memory capacity. Therefore, to put the memory blade cost into context with the memory subsystem, the cost is shown as a percentage of the total remote DRAM costs (memory blade cost divided by remote DRAM costs), using 32 GB of remote memory. Note that for clarity, the cost range on the horizontal axis refers only to the memory blade interface/packaging hardware excluding DRAM costs (the fixed DRAM costs are already factored in to the results). In other words, at 100% cost overhead the price of the memory blade (backplane interface, memory controller and channels, packaging) is *equal* to the cost of 32 GB of remote memory. Assuming commodity DDR2 memory prices, the total cost for 32 GB of remote memory is $480.

As can be seen in Figure 8, the hardware cost break-even points for `PS` and `FGRA` are high (260% and 120%, respectively) implying a sufficiently large budget envelope for the

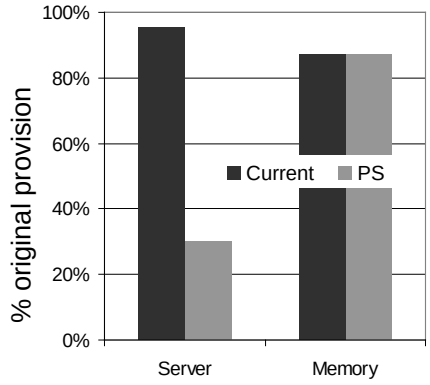


Figure 9: Consolidation improvements enabled by disaggregated memory.
Consolidation with disaggregated memory (PS) enables much greater consolidation than state-of-the-art techniques (Current).

memory blade implementation. Based on the straightforward design and limited functionality of the memory blade, it is expected that the overhead of a realistic implementation of a memory blade could be below 50% of the 32 GB remote DRAM cost, given 2010 market prices. This overhead can be reduced further by considering higher capacity memory blades; for example, the cost overhead is likely to be below 7% of the remote DRAM cost of a 256 GB memory blade.

4.4 Server Consolidation

Viewed as a key application for multi-core processors, server consolidation improves hardware resource utilization by hosting multiple virtual machines on a single physical platform. However, memory capacity is often the bottleneck to server consolidation because other resources (e.g., processor and I/O) are easier to multiplex, and the growing imbalance between processor and memory capacities exacerbates the problem. This effect is evident in the real-world *web2.0* traces, where processor utilization rates are typically below 30% (rarely over 45%) while more than 80% of memory is allocated,

indicating limited consolidation opportunities without memory expansion. To address this issue, current solutions either advocate larger SMP servers for their memory capacity or sophisticated hypervisor memory management policies to reduce workload footprints. However, these alternatives incur performance penalties, increase costs and complexity, and do not address the fundamental processor-memory imbalance.

Memory disaggregation enables new consolidation opportunities by supporting processor-independent memory expansion. With memory blades available to provide the second-level memory capacity, it is possible to reduce each workload's processor-local memory allocation to less than its total footprint (`M-max`) while still maintaining comparable performance (i.e., <3% slowdown). This workload-specific local vs. remote memory ratio determines how much memory can be freed on a compute server (and shifted onto the memory blade) to allow further consolidation. Unfortunately, it is not possible to experiment in live datacenters to determine these ratios. Instead, the typical range of local-to-remote ratios is determined using the simulated workload suite. This range is then used to investigate the potential for increased consolidation using resource utilization traces from production systems.

The consolidation benefits are evaluated using the *web2.0* workload (CPU, memory and IO resource utilization traces for 200+ servers) and a sophisticated consolidation algorithm similar to that used by Rolia et al. [78]. The algorithm performs multi-dimensional bin packing to minimize the number of servers needed for given resource requirements. The other two traces are not considered for this experiment. One real-world trace, *Animation*, is CPU-bound and runs out of CPU before it runs out of memory, so memory disaggregation does not help. However, as CPU capacity increases in the future,

there will likely be a similar situation as *web2.0*. *VM consolidation*, on the other hand, does run out of memory before it runs of out CPU, but these traces already represent the result of consolidation. In the absence of information on the prior consolidation policy, it is hard to make a fair determination of the baseline and the additional benefits from memory disaggregation over existing approaches.

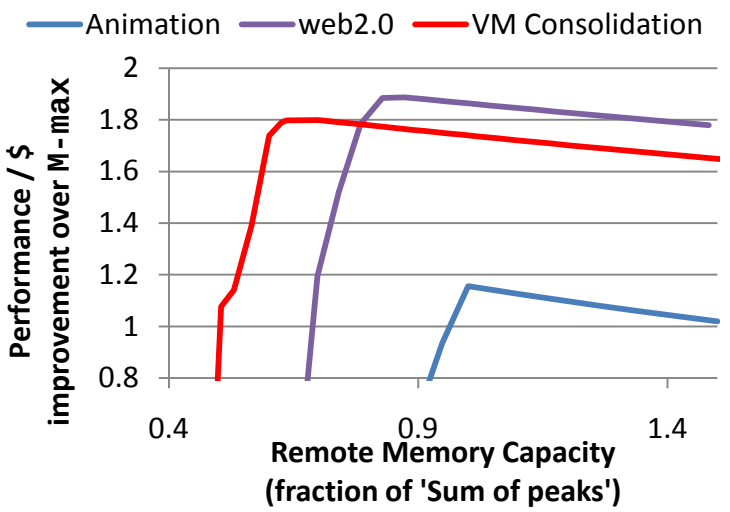
As shown in Figure 9, without memory disaggregation, the state-of-the-art algorithm ("Current") achieves only modest hardware reductions (5% processor and 13% memory) because limited memory capacity precludes further consolidation. In contrast, page-swapping–based memory disaggregation corrects the time-varying imbalance between VM memory demands and local capacity, allowing a substantial reduction of processor count by a further 68%. These results show that disaggregated memory can offer substantial improvements in VM consolidation in memory-constrained environments. This will be crucial for future data centers to enable consolidation to reduce server power and costs.

4.5 Ensemble-level Memory Sharing

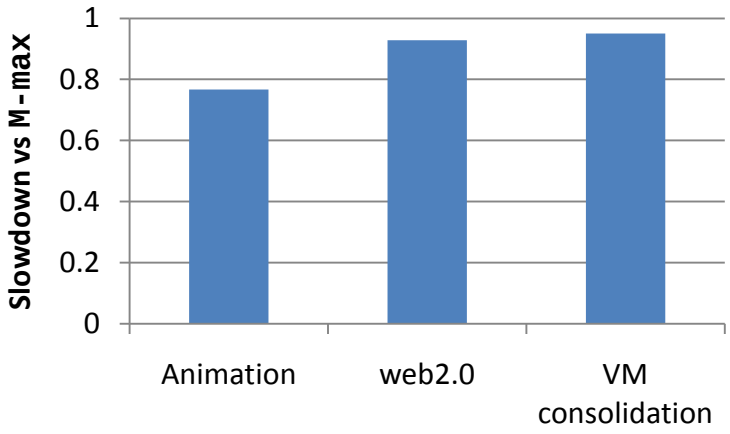
This section examines the benefits of disaggregated memory in multi-workload server ensembles with time-varying requirements. By dynamically sharing memory capacity at an ensemble level, disaggregated memory can potentially exploit the inter- and intra-workload variations in memory requirements. This variation is highlighted by the difference between the *peak of sums* and the *sum of peaks.* The *peak of sums* is the maximum total memory required across the ensemble at any single point in time. On the other hand, the *sum of peaks* is the sum of the worst-case memory requirements of all the servers based on the applications being run. In conventional environments, servers must

be provisioned for the worst-case memory usage (sum of peaks) to avoid potentially-catastrophic performance losses from underprovisioning (which may lead to swapping/thrashing). However, the peak of sums is often much smaller than the sum of peaks as servers rarely reach their peak loads simultaneously; systems provisioned for worst-case demands are nearly always underutilized. Ensemble-level sharing allows servers to instead be provisioned for the sum of peaks, saving costs and power.

The potential of ensemble-level memory blade sharing is evaluated for a 16-server blade enclosure running a mix of enterprise workloads with varying memory requirements (similar to the scenario shown in Figure 2(b)). The three real-world enterprise datacenter workload traces are examined (*Animation, VM consolidation*, and *web2.0*), and a mixed workload trace is created using the simulated workloads to mimic the same memory usage patterns. Each trace is divided into epochs and the *processing done per epoch* is measured. These rates are compared across different configurations to estimate performance benefits. To focus solely on the achievable performance by remote memory, a simple policy is assumed where, at the beginning of each epoch, each compute blade requests the additional memory it needs from the memory blade. (In a task-scheduling environment, this could be based on prior knowledge of the memory footprint of the new task that will be scheduled.) For the cost of the memory blade, the price is conservatively estimated to be approximately that of a low-end system. Based on the discussion in Section 4.3, this estimate is expected to be conservative because of the limited functionality and hardware requirements of the memory blade versus that of a general purpose server.



(a)



(b)

Figure 10: Ensemble level sharing results.

(a) Performance-per-dollar as remote memory capacity is varied. (b) Slowdown relative to per-blade worst-case provisioning (`M-max`) at cost-optimal provisioning.

Figure 10(a) shows the performance-per-memory-dollar improvement, normalized to the `M-max` baseline, for `PS` over a range of remote memory sizes. These results focus on the `PS` design because the `FGRA` design is not as competitive due to its inability to migrate frequently accessed data to local memory (see Section 4.2). Figure 10(a) shows that both the *VM consolidation* and *web2.0* traces benefit substantially from ensemble-level provisioning, gaining 78% and 87% improvement in performance-per-dollar while

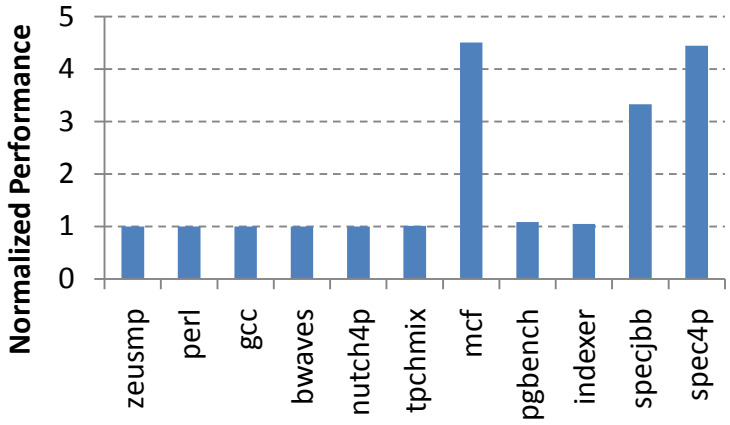


Figure 11: Alternate FGRA design that supports page movement.

Normalized performance when FGRA is supplemented by NUMA-type optimizations. Several large memory workloads significantly benefit from exploiting locality.

requiring only 70% and 85% of the sum-of-peaks memory capacity, respectively. These savings indicate that the remote memory capacity can be reduced below worst-case provisioning (sum of peaks) because demands in these workloads rarely reach their peak simultaneously. In contrast, the peak of sums closely tracks the sum of peaks in the *Animation* trace, limiting the opportunity for cost optimization.

The results indicate that a cost-optimized disaggregated memory is able to provide a performance-per-dollar improvement compared to the `M-max` baseline (worst-case provisioning). It is also important to understand the raw performance of the disaggregated memory solution relative to the `M-max` baseline to ensure the performance obtained is at an acceptable level. Figure 10(b) shows the performance sacrificed by the per-workload cost-optimal design (as determined by the performance-per-dollar peak for each workload in Figure 10(a)). There is minimal performance loss for the *web2.0* and *VM consolidation* traces (5% and 8%), indicating that disaggregated memory can significantly improve cost-efficiency without adversely affecting performance. For the *Animation* traces there is

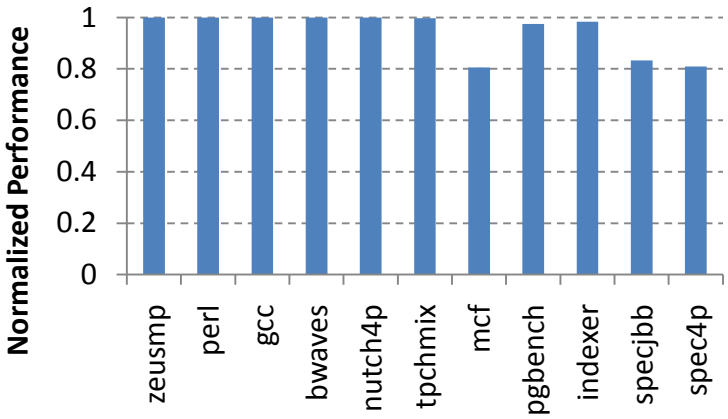


Figure 12: Alternate FGRA designs that tunnels accesses over PCIe.

Normalized performance loss from tunneling FGRA accesses over a commodity interconnect. Performance decreases by at most 20%.

a larger performance penalty (24%) due to its consistently high memory demands. Compared to the M-median baseline, the disaggregated memory designs show substantial throughput improvements (34-277X) for all the traces.

4.6 Alternate Designs

4.6.1 FGRA with Page Migration

As discussed earlier, the FGRA design suffers relative to PS because it does not exploit locality by swapping heavily used remote pages to local memory. This disadvantage can be addressed by adding page migration to FGRA, similar to existing CC-NUMA optimizations (e.g., Linux's memory placement optimizations [18]). To study the potential impact of this enhancement, a hypothetical system was modeled that tracks page usage and swaps the most highly used pages into local memory at 10 ms intervals. Figure 11 summarizes the speedup of this system over the base FGRA design for M-median compute blades. For the high-footprint workloads that exhibit the worst performance with FGRA (*mcf, SPECjbb, and SPEC4p*), page migration achieves 3.3-4.5X improvement,

with smaller (5-8%) benefit on other high-footprint workloads. For all workloads, the optimized FGRA performs similarly to, and in a few cases better than, PS. These results motivate further examination of data placement policies for FGRA.

4.6.2 FGRA Tunneled Over PCIe

The hardware cost of FGRA can be reduced by using a standard PCIe backplane (as in the PS design) rather than a coherent interconnect, as discussed in Section 3.3.2. This change incurs a latency and bandwidth penalty as the standardized PCIe interconnect is less aggressive than a more specialized interconnect such as cHT. Figure 12 shows the change in performance relative to the baseline FGRA. Performance is comparable, decreasing by at most 20% on the higher memory usage workloads. This performance loss may be acceptable if the costs of extending a high-performance interconnect like cHT across the enclosure backplane is high.

4.6.3 Simulated Hypervisor Sensitivity Study

The sensitivity of these results to the VMM overhead and memory latency parameters in Table 2 was also studied. For a range of conservative values, there was no qualitative difference in the results (less than 2% performance difference), so these sensitivity results are omitted for brevity and clarity.

4.7 Discussion

4.7.1 Simulation Methodology

There are several options for evaluating the memory architectures, including trace-based simulation, detailed execution-driven simulation, and software or hardware

implementation. The end goal of the evaluation is to determine the performance of the two designs on a variety of large memory workloads that may touch several gigabytes of memory (which can take up to several hours of real-world time). Thus the key requirements for the evaluation are the ability to: (1) flexibly model designs with different interconnect types and software interactions, (2) quickly achieve performance numbers so that multiple large workloads can be tested, and (3) be developed in a reasonable amount of time to expedite the memory blade research.

Trace-based simulation, which processes main-memory access traces to estimate the overall performance of a design, is the only option that allows all three requirements to be met. Although this method does not provide the most accurate model of the memory designs, it allows flexibility in the parameters modeled, can quickly process large amounts of memory-traces, and can be developed quickly due to its straightforward nature. Detailed execution-driven simulation offers accurate and flexible models, but has significant slowdown compared to real-world run-times due to the details that are modeled. A software implementation may be inadequate to model some of the fine-grained design details, such as transactions on a cache-block level, and a hardware implementation may take a significant amount of time to develop. However, these options have their benefits in providing greater detail than is possible with simulation. Chapter 5 examines a prototype of a software implementation that is developed to greater understand the software-implications of disaggregated memory.

4.7.2 Evaluation Assumptions

The trace-based simulations do not model interconnect routing, arbitration, buffering, and QoS management in detail. If interconnect utilization is not near saturation, these

omissions will not significantly impact transfer latencies. The results of the simulations confirm that per-blade interconnect bandwidth consumption falls well below the capabilities of PCIe and HT. However, the number of channels to the memory blade may need to be scaled with the number of supported clients.

Although this modeling forgoes the ability to model overlap between processor execution and remote memory accesses with trace-based simulations, the memory reference traces are originally collected from a simulator that does model overlap of local memory accesses. Additionally, the overlap for remote accesses is likely to be negligible due to the relatively high latencies to the remote memory blade. The trace simulation methodology is unable to model the actual content of memory because the size of the traces would be intractable if all updates to memory content were included. Thus the simulator models the location of pages in the memory hierarchy, but not their actual content. Despite these limitations, because the focus of the evaluation in this chapter is on the performance of the memory blade, these drawbacks are minimal.

4.7.3 Impact of the Memory Blade on Ensemble Manageability

Memory disaggregation has both positive and negative impacts on enterprise system reliability, availability, security, and manageability. From a reliability perspective, dynamic reprovisioning provides an inexpensive means to equip servers with hot-spare DRAM; in the event of a DIMM failure anywhere in the ensemble, memory can be remapped and capacity reassigned to replace the lost DIMM. However, the memory blade also introduces additional failure modes that impact multiple servers. A complete memory-blade failure might impact several blades, but this possibility can be mitigated by adding redundancy to the blade's memory controller. It is likely that high availability

could be achieved at a relatively low cost, given the controller's limited functionality. To provide security and isolation, the memory blade design enforces strict assignment of capacity to specific blades, prohibits sharing, and can optionally erase memory content prior to reallocation to ensure confidentiality. From a manageability perspective, disaggregation allows management software to provision memory capacity across blades, reducing the need to physically relocate DIMMs.

4.8 Summary

A detailed, simulation-based evaluation of disaggregated memory shows that it effectively provides expanded memory capacity and memory sharing, enabling significant performance and cost benefits. This expanded capacity can be leveraged to enable greater levels of consolidation, removing memory capacity as a bottleneck. The two proposed memory architectures for accessing the memory blade, PS and FGRA, each provide good performance but have interesting tradeoffs. In particular, it was expected that the FGRA design would outperform the PS design due to its higher speed, lower latency interconnect and fine grained access. However, the swapping of remote data to local memory proved to be much more important to overall performance. This result was confirmed by evaluations of alternate FGRA designs that supported such data movement indicated the importance of those policies.

Chapter 5

Hypervisor Prototype of Disaggregated Memory

The simulation-based evaluations discussed in the previous chapter are crucial for providing good performance estimates of a range of disaggregated memory designs. However, simulator models may not fully match existing hardware, and the slowdown or abstraction of detail associated with simulation may prevent an understanding of the system-level implications of the new memory architectures. To gain a better understanding of such factors as the extent of software-stack changes required and hardware-software interactions, a software prototype is developed for this thesis that implements the primary functionality of disaggregated memory. The PS design is used as the basis of the prototype because the required changes are all in software, specifically within the hypervisor. This chapter details the work done in prototyping the PS design by modifying the Xen hypervisor to support detecting, accessing, and managing remote memory.

Section 5.1 describes the motivation behind the development of a prototype, detailing some of the benefits it provides. Section 5.2 describes the design decisions that influence the prototype. Section 5.3 details the changes made to the hypervisor to implement disaggregated memory, and in Section 5.4 the prototype is evaluated and used to verify

some of the results obtained through simulation. Section 5.5 discusses some of the unique lessons learned from doing the implementation.

5.1 Motivation

A real-world prototype of disaggregated memory, either in hardware or software, can serve to complement the findings that are obtained through simulation by providing a different level of implementation detail. The simulation methodology described in Chapter 4 allows for very flexible analysis of the two disaggregated memory architectures, but has some limitations in illustrating the system-level interactions of the designs. Our trace-based simulator does not actively simulate the operating system or detailed hardware; these trade-offs were made in favor of faster run times. Thus a disaggregated memory prototype can complement simulation and help to identify the full extent of the modifications required at the hardware- or software-level. It can also enable full-scale, large-memory workloads to be run in a reasonable amount of time without requiring the abstraction trade-offs of trace-based simulation. This execution speed is essential for allowing a wide variety of workloads to be evaluated.

A prototype can also offer insights into architectural interactions with existing hardware. Modeling these hardware interactions in simulation would require a potentially lengthy development cycle to create and validate the hardware models, and would have substantial slowdown compared to real world machines. For example, our simulator does not model the architectural behavior of the hardware virtualization support that has been added to recent x86 processors; this architectural behavior varies across different processor models, and only some details are covered in recent publications, e.g., [14]. Because both disaggregated memory designs utilize virtualization, the impact of

hardware virtualization support on the memory designs is not apparent through simulation. Thus the development of a prototype can further serve to help identify interesting interactions between the new memory architectures and real-world hardware. Additionally, it can serve as an important validation for simulations to ensure that, to a first order, the primary aspects that influence performance are modeled.

Another motivation for the development of a prototype of disaggregated memory stems from certain insufficiencies of the trace-based simulations used in this work. One drawback to the trace-based simulation methodology is the inability to track the contents of memory. Because the traces record billions of accesses to multiple gigabytes of memory, the size of the traces would become unwieldy if they were to include the contents of each memory access. While this limitation does not preclude performance studies on multi-level memory hierarchies, it does prevent studies of designs where the memory content is important, such as content-based page sharing at the memory blade level. As opposed to simulation, a prototype can properly update the contents of memory, enabling experiments that need memory contents to be faithfully tracked. For all of these reasons, a prototype of disaggregated memory is developed for this thesis to provide a more complete understanding of the new memory architectures

5.2 Prototype Design Choices

One of the primary decisions in developing a prototype is whether it should be implemented in hardware, software, or a combination of the two. In this thesis, a software-based prototype of disaggregated memory is developed by modifying a hypervisor to support remote memory. A software-based prototype is chosen because of the availability of open-source hypervisors, the established hypervisor development

community, and the ability to run the prototype on multiple different machines, if needed. A hardware prototype is not considered due to money and time constraints. Building the disaggregated memory hardware—the memory blade and the interconnection hardware—could be expensive due to the custom components, and the prototype may potentially not flexible enough to test a wide variety of parameters and configurations. Additionally, a hardware prototype would likely require software modifications to support accessing the remote memory, and the implementation of both hardware and software infrastructures would lead to lengthy development times.

Based on the decision to develop a software-based prototype, the PS design is chosen for implementation because it primarily requires software changes to existing systems. However, the PS design has certain aspects that are not amenable to being implemented in a software prototype. Therefore it is important to discuss first, the primary elements of the design, and second, how those elements will be reproduced in a software-based prototype.

The primary hardware and software components of the PS disaggregated memory architecture include:

1. A memory blade that provides remote memory capacity to multiple compute servers, connected via a commodity interconnect.
2. Hypervisor support for detecting accesses to remote regions, obtaining remote pages, and evicting local pages.
3. Hypervisor paging or balloon driver support for dynamic memory capacity reallocation.

In a software-based prototype, the memory blade, the interconnect, and its remote capacity do not exist and must instead be emulated. The memory blade and interconnect are emulated through small routines in the hypervisor that implement their general behavior and roughly estimate their performance. The remote capacity is emulated by dividing up the physical machine's capacity into "local" and "remote" regions, and adding in small delays to emulate remote access latencies. The method for tracking these regions is discussed in further detail below. In a real implementation, the memory blade would be connected to multiple compute blades. To avoid the complexity of having multiple systems communicating over an interconnect, a single, multi-socket system can instead act as both "compute blades" and the "memory blade" by treating each CPU socket as an individual "blade." The hypervisor running on the system would then act as a global supervisor of the compute and memory "blades," handling memory allocation policies. (Although this multi-socket division capability is not currently present, the prototype has been designed in a way that supports its implementation.)

To address the second component, modifications are made to an open-source hypervisor to support detecting and handling accesses to regions of remote memory. Functionality is added to the hypervisor to support a system memory address space that is made up of memory local to the system and memory that is remote. Additionally the hypervisor is modified to implement the page-swapping functionality of the PS design. The full details of these modifications are described in the next section.

The final component, support for dynamic memory capacity reallocation, can be implemented through appropriate modification of the hypervisor or balloon driver. However, this feature is complex and dependent upon a working prototype of

disaggregated memory that supports multiple clients. Because of time constraints, dynamic memory capacity reallocation is not currently supported, but is planned as future work.

5.3 Hypervisor Modifications

Based on the requirements for the prototype PS design, the Xen hypervisor [10] was chosen to be modified to support disaggregated memory. Xen is a well-established open-source virtualization software system that has been under development for several years. It offers a high-quality, well-documented hypervisor, and strong development community support. The primary functionality that must be added to the hypervisor is the ability to detect and handle accesses to remote memory. The PS design assumes this support is added by extending the hypervisor's existing mapping of guest virtual addresses to system machine addresses to include remote memory addresses. However, the Xen hypervisor has three different modes of address translation: *paravirtualization*, *shadow page tables*, and *hardware assisted paging*. It would be impractical to completely implement disaggregated memory for all of the translation modes because of the development time that would be required. These modes are described in greater detail below to motivate the implementation choice of *hardware assisted paging*.

5.3.1 Paravirtualization

By default, Xen uses paravirtualization, in which the operating system is modified to use specialized system calls to facilitate virtualization and improve performance. These specialized system calls include calls to have the hypervisor help set up guest to system mappings. However, paravirtualization is more limited than traditional virtualization as it

requires modifications to the operating system, and thus the cooperation of OS developers. This cooperation may not always be available; Microsoft is unlikely to add Xen-compatible paravirtualization support to their operating systems given that they have a competing hypervisor. More importantly, one of the main goals of the PS design is to be transparent to the OS. Paravirtualization cannot provide this transparency, and therefore is not used for this prototype.

5.3.2 Shadow Page Tables

In addition to paravirtualization, Xen supports hardware-assisted virtualization by using recent virtualization extensions that have been released by AMD and Intel. In this case, Xen can run guest operating systems with no modifications, and instead uses "shadow" page tables to track the mappings from the guest OS to the system machine addresses. The hypervisor uses shadow page tables track the guest OS's updates to its page tables, intervening where necessary and installing the proper mapping in the hardware page table. However, this scheme has a negative performance impact compared to paravirtualization due to the tracking and trapping that is necessary to keep the mappings up to date; this performance loss is the motivation behind the hardware assisted paging mode. Although the prototype is implemented using hardware assisted paging, for completeness and to demonstrate the feasibility of the PS design, the steps needed to support disaggregated memory using shadow page tables are described below.

Using shadow page tables would be a fairly direct implementation for the PS modifications. Because the hypervisor is actively involved in the translation of guest to host addresses, it would be a simple extension to modify the hypervisor routine that handles page table updates to also support local and remote memory regions. By doing

so, any remote regions of memory can have their permissions removed, which will generate a page fault when they are accessed. This scheme follows the mechanism described in Section 3.3.1 which uses page faults to trap into the hypervisor and allow the hypervisor to handle accessing the remote page. However, care must be taken to use efficient mappings for the remote regions because each process has its own page table, resulting in multiple shadow page tables per operating system.

5.3.3 Hardware Assisted Paging

The most recent processors from AMD and Intel support “hardware assisted paging” (HAP), which sets up a second set of page tables in hardware that handle the translation from a guest physical address to a system machine address [14]. By doing this translation in hardware, the hypervisor not only supports unmodified operating systems, but also no longer has to intervene on most page table updates once the initial guest to host mappings are created. Trapping frequently to the hypervisor is one of the main sources of performance slowdown when using shadow page tables. HAP alleviates this problem by avoiding many of those calls into the hypervisor, and having architectural support for caching full translations from guest OS to system machine address. Furthermore, only one nested page table is required per guest OS, potentially saving memory space over the shadow page table approach.

Because of these benefits, the implementation of disaggregated memory is done in HAP mode for the present study. A block diagram of the code modifications and functionality added is shown in Figure 13. The main method for detecting accesses to remote pages, specifically by marking those remote pages as “not present” in their page table entries, remains the same. These markings are set up during an initialization phase

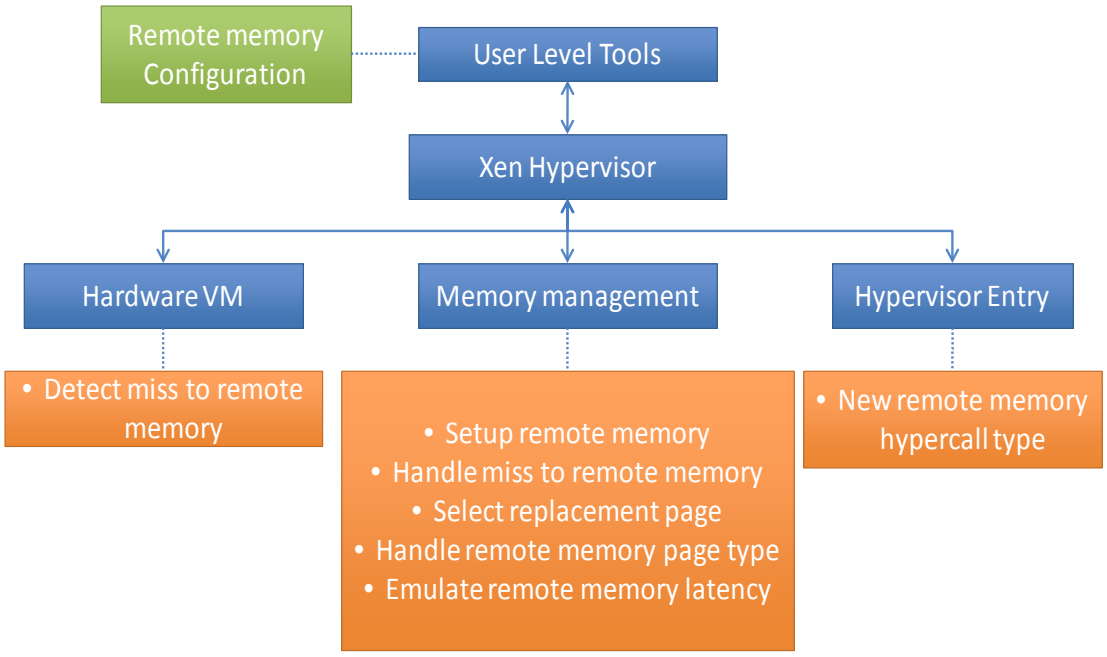


Figure 13: Block level diagram of prototype software changes.
Blue blocks represent major software components, green and orange blocks indicate new and modified source code files, respectively. A majority of the required code changes are within the memory management portions of the Xen hypervisor.

that configures the local and remote regions. If a page that is marked as "not present" is accessed, then the hypervisor must intervene. This action is modified compared to the normal activity in HAP mode. Upon such a page fault occurring, the hypervisor must check the address and the page type to see if the faulting address is a remote page. If it is a remote page, a handler is called to locate the remote page, access it, choose and swap a local page with the remote page, and update the mappings for the two pages to indicate their now local and remote statuses, respectively. Although this design requires trapping to the hypervisor, the overall performance is significantly faster than if the data must be accessed from disk.

Remote pages are indicated by being marked as "not present" in their page table entries, and a per-domain bitmap that indicates the local/remote status for all of the guest

physical addresses. In a full implementation of disaggregated memory, this per-domain bitmap is not explicitly needed; marking the remote pages as “not present” and doing a base and bounds check on the system machine address would indicate if the address is located on remote memory. However, for the purposes of this prototype, the bitmap provides multiple benefits—easier implementation by avoiding the need for separate local and remote memory allocators; easily changeable local/remote allocations as VMs are started and destroyed; and a simple method for picking replacement pages. Currently a round-robin policy is used to choose the local page to be evicted upon remote access by simply scanning through the bitmap to find the next local page. (This scheme will find the next local guest physical address.) This replacement policy is simplistic, and thus likely does not offer optimal performance. However, it provides very fast selection times, minimizing the time spent in the hypervisor.

In order to support the remote memory infrastructure, several initialization and setup functions were added, and user level tools were developed to control this code. These functions include setting up domains to have a certain percentage of local versus remote memory, and obtaining performance counters such as number of remote pages accessed. As shown in Figure 13, all of the modifications discussed in this section are primarily to the memory management code of Xen, indicating that the implementation changes needed are relatively localized.

5.3.4 Implementation Limitations

In prototyping disaggregated memory in the Xen hypervisor, some functionality had to be omitted, and there are some limitations to the prototype. One of the biggest changes is the absence of a physical memory blade that provides remote capacity. As discussed

earlier, remote capacity is emulated by dividing up a machine's capacity into "local" and "remote" regions. This emulation has a few drawbacks: first, it does not allow expansion of the memory capacity of the system; second, the emulation is unable to include the interactions that would be required to generate PCIe requests and send them to the memory blade; and last, power savings and accurate performance numbers cannot be measured from the prototype.

The first drawback primarily affects large consolidation experiments and does not affect many of the workloads that can be set up with reasonable effort. If needed, workloads can be configured to have a restricted local memory to mimic conditions of memory-constrained environments, similar to the experiments in Section 4.2. The second and third drawbacks suggest further study of a memory blade hardware prototype. Another alternative is to use a second compute blade and connect it to the test system using PCIe. In this manner, the second compute blade would mimic a memory blade.

The current prototype does include coarse modeling of PCIe latency overheads by adding in some additional latency to "remote" memory accesses. The prototype emulates "remote" memory by adding in a call to sleep for a few microseconds, which is configurable at run time. There are also configurable parameters to set remote memory bandwidth. However, this coarse modeling is not as robust as changing the parameters in simulation; sleeping on the scale of microseconds is imprecise for non-realtime operating systems due to timer granularity, scheduling windows, and interrupts [98].

It is important to note that this software-based prototype of disaggregated memory is not necessarily indicative of the absolute performance attainable. There are certain aspects of the implementation that are unoptimized, or are designed for ease of

implementation rather than high performance. Additionally, other important aspects such as the PCIe interconnect are not present. Thus the performance of the disaggregated memory prototype is likely representative of the overall trends in performance, but should not be taken as absolute numbers without further study.

5.4 Evaluation and Validation of Simulation Results

The Xen-based implementation of disaggregated memory, along with code to emulate the slower performance of remote memory, is used to run several microbenchmarks and SPEC CPU 2005 benchmarks. These tests validate both the proof of concept of disaggregated memory and, at a high level, the simulation results of Chapter 4. All tests are done on a dual-socket machine with two AMD Opteron 2354 Quad Core processors, providing a total of 8 cores. The machine has 32 GB of memory and has the latest generation of hardware virtualization technology, including HAP support. The Xen code modified is based on the Xen 3.4.1 unstable source tree. All VMs are run on an unmodified Debian Linux 2.6.18.

First a custom microbenchmark is used to determine the average access time for a remote page using the Xen-based prototype. The microbenchmark walks through memory at a 4 KB page stride, and records the total time to walk the entirety of memory. For these runs the VM images were configured to have 1 GB of memory; additional tests with larger memory capacities were quantitatively very similar. In order to calculate remote memory access time, a percentage of the VM's memory is set to be remote (20%, 40%, or 60%), the parameter for remote memory access time is set (0, 4, 10, 15 or 20 µsec), and the overall run time for the microbenchmark is measured. The run time is

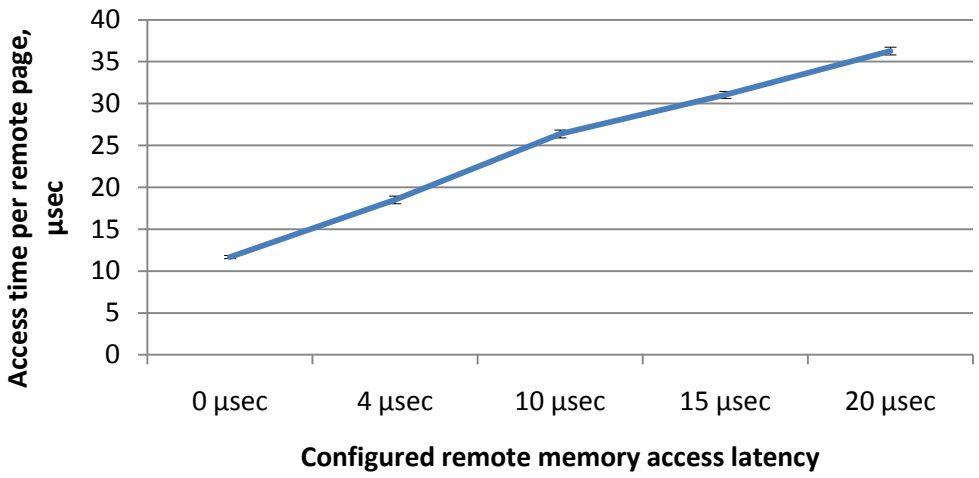


Figure 14: Average access time per remote page

The value set for remote memory access latency is varied. Note that the increases in latency from 0 to 4 µs, and from 4 to 10 µs are higher than expected.

compared to a baseline run with no remote memory, the difference is calculated, and the result is divided by the total number of remote pages that were touched.

The results shown in Figure 14 indicate the average access time per remote page, as the defined remote memory access latency is varied. The results shown are for 40% of memory being set to remote. When remote memory latency is set to 0 µsec, or no overhead, there is a baseline 12 µsec penalty for the hypervisor overhead of detecting the remote page, selecting a local page to evict, and updating the mappings. This overhead is an order of magnitude higher than the overhead previously estimated in Section 4.1 (330 ns), which was derived from prior work [79]. Several sources for this discrepancy are likely. First, the detection of remote memory pages and the access path for accessing the remote pages is not highly optimized, and could be sped up with some code improvement. Second, the microprocessor architecture changes to support HAP may be increasing the time required to detect that a page is not present. To provide HAP, the processor must do several page table walks in order to translate the guest address to the

system address. Without HAP, this translation would only require a single page table walk. The multiple page table walks increases the time to resolve the page table entry. Previously published work estimates that these nested walks are 3.9-4.6 times slower than native performance [14]. This performance is likely to improve in the future as AMD and Intel optimize their virtualization support, which will help bring the remote memory access latency back to the originally estimated numbers.

Figure 14 also shows the impact of changing the remote memory access latency. The implementation has difficulties in properly emulating smaller latencies. Increasing the latency from 0 µsec to 4 µsec results in a 6 µsec increase in overall access latency. Increasing from 4 to 10 µsec increases total latency 8 µsec, while scaling from 10 to 15 µsec and 15 to 20 µsec increases latency the expected 5 µsec each time. As discussed in Section 5.3.4, this inability to have very fine-grained controls of emulated latencies is one of the main reasons why this implementation serves as a supplement to simulation, but cannot replace the results of simulation. However, the overall latencies are consistent enough to be useful for performance estimation.

The Xen-based prototype of disaggregated memory was used to run the SPEC CPU benchmarks used in the evaluations in Chapter 4 (*perl*, *gcc*, *bwaves*, *mcf*, and *zeusmp*). Here, each VM is set up with 2 GB of memory, and only one VM is run at a time. Figure 15 presents the normalized run time of each benchmark when varying the total percentage of remote memory from 0% (0 GB) through 95% (1.9 GB). The results show that disaggregated memory is able to provide high performance for most workloads, despite a large portion of memory being remote. Excluding *mcf*, all other benchmarks suffer less than 20% slowdown when up to 80% of memory is remote.

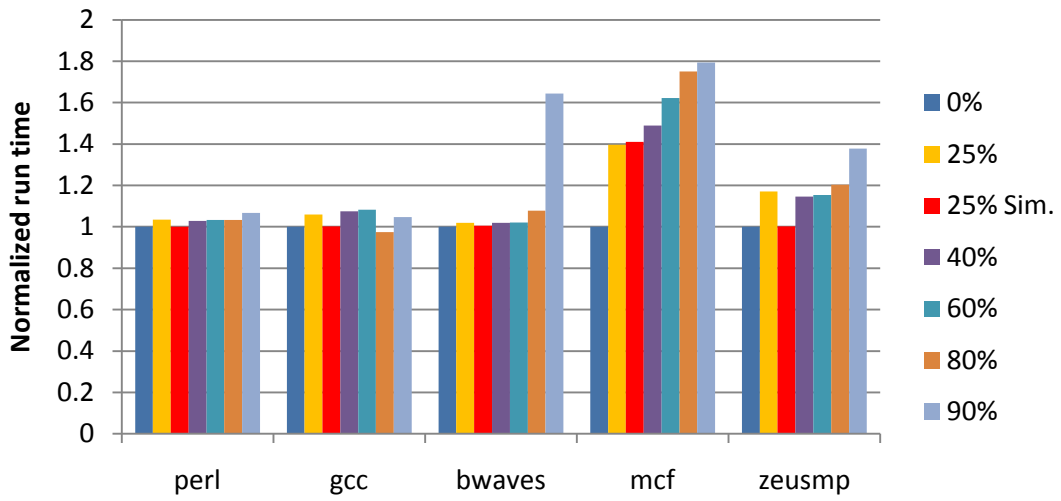


Figure 15: Performance SPEC CPU benchmarks with varying percentages of remote memory.

Virtual machines are configured with 2 GB of memory. Four workloads can have up to 80% of memory be remote with less than 20% slowdown. Red bars show simulation results for 25% remote, using hypervisor trap latencies based on the prototype.

The results with 25% of memory being remote strongly correlate with the simulation results for the `M-median` configuration with disaggregated memory for all workloads other than *zeusmp*. Note that these simulations were tuned to use the hypervisor trap time (~12 µs) based on the prototype. These results provide high level validation for the simulation results and confidence in the simulation and prototype.

5.5 Lessons Learned

The implementation of disaggregated memory support in Xen has provided several interesting lessons. As mentioned above, the assumed simulation parameter for hypervisor trap time was significantly affected by new architectural behavior due to virtualization support. The changes to support HAP result in up to 4 times longer page table walks. This slower table walk is one factor that causes the 12 µs hypervisor trap

time of the prototype, as opposed to the expected 330 ns latency. This latency gap indicates the need for further optimization that is aware of the nested page table walks. If architectural support is available, a cache of addresses of recently evicted pages may help to speed up remote access detection. Without requiring architectural support, one possibility for a future study is to operate in shadow page table mode, which would not require the lengthy path to determine a page fault.

Despite the higher trap latency, the prototype proved useful for evaluating performance and validating the results of Chapter 4. Across the 5 workloads examined, for a VM with 2 GB of memory, all but *mcf* were able to tolerate up to 80% of memory as being remote with less than 20% performance degradation. Additionally, when the simulator's parameters were tuned to match the 12 µs hypervisor trap time, the trends were in strong agreement with the prototype for all workloads but *zeusmp*.

The prototype development has shown that the majority of the code that needs modification resides in the memory management portion of the Xen source code. This finding is encouraging as it implies that code changes required for disaggregated memory will be localized. However, there are some complications that will require careful modification of outside code. Initially the prototype was developed with remote memory pages having their own "remote" page type in Xen. Having a unique page type required modification of several other portions of code, including the I/O management, to make the code understand a "remote" page type being a valid RAM page. Instead of pursuing these changes, remote pages were left as normal RAM pages, and were tracked by being marked as "not present" in the page table and in a per-domain bitmap. This tracking required far less code modification, and kept the modifications localized to the memory

management code. However, it is likely that an eventual implementation of disaggregated memory will require a unique page type for remote pages, and therefore a more extensive set of code changes.

Finally, initial experiments with the custom microbenchmark to measure remote access time indicated a large variability in latency, up to an order of magnitude difference. Upon deeper inspection, the source of large variability was shown to be the hypervisor maintaining superpages in the guest page tables. When a local page was selected for eviction that was within the superpage, the hypervisor would be forced to break up the superpage and provide mappings for all of the individual pages. This additional mapping activity increased the overall latency by an order of magnitude due to the time consuming operations. Thus to avoid the overhead of breaking superpage mappings, for this study superpages are broken up upon initialization. However, this methodology has a performance impact on certain workloads such as *mcf*, decreasing performance by approximately 15%, compared to the baseline. Although the impact was much more minor on other workloads (1-3%), care must be taken in a production-quality implementation of disaggregated memory to support the existence of superpages while maintaining high performance on remote memory accesses.

Chapter 6

Extending Disaggregation

Disaggregated memory serves as one design point for disaggregation, showing the potential of an architecture that intelligently reorganizes memory resources to promote efficiency. In this chapter, disaggregation is applied at the compute level to create new server architectures through the combination of memory and compute disaggregation, packaging and cooling optimizations, and flash-based disk caching. Starting from a broader perspective than individual servers, these architectures are targeted to specifically address the performance, power, and cost efficiency needs of large-scale data centers. These data centers, driven by the growth of the internet sector, have unique needs due to the thousands of required servers. Thus there is a strong necessity for architectures that can provide the compute and memory capacity needed by internet sector workloads at drastically lower costs than traditional servers. Disaggregation at the compute and memory level can help to achieve that needed efficiency.

Section 6.1 discusses trends in large-scale, “warehouse computing” data centers that require new, efficient server architectures. This is a new area of study for architects, and thus Section 6.2 describes the evaluation environment that was developed in order to study this new class of workloads. The models and metrics used to evaluate warehouse computing environments are explained in Section 6.3; in these environments the total

performance per dollar of the data center is perhaps even more important than the raw performance. Section 6.4 details the individual components of the proposed new server architectures and evaluates them across the workloads. In Section 6.5 the whole unified architecture is evaluated. Section 6.6 discusses some alternatives and future work, and Section 6.7 summarizes this chapter.

6.1 Motivation and Contributions

In the previous chapters, disaggregation at a memory resource level has been proposed to increase the efficiency of servers. Looking beyond memory capacity, there are important market trends at an even bigger scale. Recent market data identifies the "internet sector" as the fastest growing segment of the overall server market, growing by 40-65% every year, and accounting for more than 65% of low-end server revenue growth in 2007. Indeed, several recent keynotes [11, 17, 49, 65, 68, 92] are indicative of the growing importance of this area.

However, the design of servers for this market poses several challenges. Internet sector infrastructures have millions of users, often running on hundreds of thousands of servers, and consequently scale-out is a key design constraint. These infrastructures have been likened to the design of a large warehouse-style computer [11], with the programs being distributed applications like mail, search, etc. At this scale, the data center infrastructure is often the largest capital and operating expense. Additionally, power and cooling make up a significant fraction of the operating costs.

These constraints, in turn, lead to several design decisions specific to internet sector infrastructures. The focus on costs motivates leveraging the "sweet spot" of commodity pricing and energy efficiency, and also reflects in decisions to move high-end hardware

features into the application stack (e.g., high-availability, manageability). Additionally, the high volume in this market and the relative dominance of a few key players – e.g., Google, MSN (Microsoft), Amazon, Facebook, Yahoo! – allow for exploring options like custom-designed servers in "green-field" datacenters built from scratch. Indeed, Google and MSN's purchase of real estate near the internet backbone or power grid for this purpose has received a lot of recent press [49].

All these trends motivate the need for research in understanding these workloads, and on new system architectures with compelling cost/performance advantages for this market. In particular, this thesis makes the following contributions. First, a detailed evaluation infrastructure is developed including a novel benchmark suite for warehouse-computing workloads, along with detailed performance, cost, and power models and metrics. Second, using these tools, four key areas for improvement are identified: CPU, packaging, memory, and disk. Third, new server architectures are designed that takes a holistic approach at addressing these bottlenecks.

Using concepts introduced in the memory disaggregation portions of this work, the disaggregation principles are extended to include compute disaggregation, and are applied at the data center level. This thesis explores disaggregating compute resources from traditional, monolithic servers and instead allocating these compute resources through smaller, more efficient systems that use low-cost, low-power components from the high-volume embedded/mobile space. This architecture is combined with memory disaggregation, novel packaging solutions, and flash-based disk caching to create a new server architecture for warehouse computing environments. The results are promising, combining to provide a two-fold improvement in performance-per-dollar for the

benchmark suite. More importantly, they point to the strong potential of cost and resource-efficient disaggregation for this class of workloads.

6.2 A Benchmark Suite for the Internet Sector

6.2.1 Key Challenges

In order to study new architectures for warehouse-computing environments, it is necessary to have representative workloads that can be used to evaluate the impact of different designs. However, a key challenge in this space is the lack of access to internet-sector workloads. The proprietary nature and the large scale of deployment are major impediments in duplicating these environments. These environments have a strong focus on cost and power efficiency, but there are currently no complete system-level cost or power models publicly available, further exacerbating the difficulties. As a result of these challenges, a new benchmark suite is developed for this work that encompasses several emerging classes of internet sector workloads, including web search, web mail, video hosting, and MapReduce-style computation.

This benchmark suite was developed to capture the aspects of “web 2.0” applications that are driving the internet sector. Many of these are described by O’Reilly [66], and include user generated content, data as a focal point, the web as a platform, and rich user interface. Another important aspect is unstructured data (e.g., web pages or user generated content), which is a core business component to internet companies such as Yahoo! or Google. One key property of many of these workloads is that the user is interacting directly with the services, and expects prompt response time to foster that interactivity. Thus for three of the workloads, Quality of Service (QoS) guarantees are

included that state a specific percentage of queries will be satisfied in a certain amount of time. Given the nature of the workloads and the internet sector, this timeframe is usually on the millisecond timescale. All of these “web 2.0” aspects are important to creating a robust benchmark suite that approximates internet sector workloads.

6.2.2 The Warehouse-Computing Benchmark Suite

The new benchmark suite created for this work has four workloads representative of the different services in internet sector data centers. Table 3 gives a summary of each benchmark and their performance metrics, and the benchmarks are discussed in more detail below.

Websearch: This benchmark is chosen to be representative of *unstructured data processing* in internet sector workloads. The goal is to service requests to search large amounts of data within sub-seconds. This benchmark uses the Nutch search engine [52] running on the Tomcat application server and Apache web server. The benchmark uses a 20GB dataset with a 1.3GB index of parts of www.dmoz.org and Wikipedia. The keywords in the queries are based on a Zipf distribution of the frequency of indexed words, and the number of keywords is based on observed real-world query patterns [97]. The performance is measured as the number of requests per second (RPS) for a Quality of Service (QoS) guarantee that greater than 95% of all queries take less than 500 milliseconds. This benchmark emphasizes high throughput with reasonable amounts of data processing per request.

Webmail: This benchmark seeks to represent *interactive internet services* seen in web2.0 applications. It uses PHP-based SquirrelMail server running on top of Apache. The IMAP and SMTP servers are installed on a separate machine using Courier-IMAP

Workload	Emphasis	Description	Performance metric
websearch	the role of unstructured data	Open source Nutch-0.9, Tomcat 6 with clustering, and Apache2. 1.3GB index corresponding to 1.3 million indexed documents, 25% of index terms cached in memory. 2GB Java heap size. QoS requires >95% queries take <0.5 seconds.	Request-per-sec (RPS) w/ Quality-of-Service (QoS)
webmail	interactive internet services	Squirrelmail v1.4.9 with Apache2 and PHP4, Courier-IMAP v4.2 and Exim4.5. 1000 virtual users with 7GB of mail stored. Email/attachment sizes and usage patterns modeled after MS Exchange 2003 LoadSim heavy users. QoS requires >95% requests take <0.8 second.	RPS w/ QoS
ytube	the use of rich media	Modified SPECweb2005 Support workload with Youtube traffic characteristics. Apache2/Tomcat6 with Rock httpd server.	RPS w/ QoS
mapreduce	web as a platform	Hadoop v0.14 with 4 threads per CPU and 1.5GB Java heap size. Two workloads are studied - distributed file write (mapred-wr) and word count (mapred-wc)	Execution time

Table 3: Details of the new benchmark suite representing internet sector workloads.

and exim. The clients interact with the servers in sessions, each consisting of a sequence of actions (e.g., login, read email and attachments, reply/forward/delete/move, compose and send). The size distributions are based on statistics collected internally within the University of Michigan, and the client actions are modeled after MS Exchange Server LoadSim "heavy-usage" profile [91]. Performance is measured as the number of RPS for a QoS guarantee that 95% of all requests take less than 800 milliseconds. The benchmark includes a significant amount of network activity to interact with the backend server.

Ytube: This benchmark is representative of web 2.0 trends of using *rich media types* and models media servers servicing requests for video files. The benchmark consists of a heavily modified SPECweb2005 Support workload, driven with Youtube™ traffic characteristics observed in edge servers by Gill, et al. [38]. The pages, files, and

download sizes are all modified to reflect the distributions seen in that work, and the QoS requirement is extended to model streaming behavior. Usage patterns are modeled after a Zipf distribution. The performance is measured as the number of requests per second, while ensuring that the QoS violations are similar across runs. The workload behavior is predominantly IO-bounded.

Mapreduce: This benchmark is representative of workloads that use the *web as a platform.* It models a cluster running offline batch jobs of the kind amenable to the MapReduce [24] style of computation, consisting of a series of "map" and "reduce" functions performed on key/value pairs stored in a distributed file system. The benchmarks use the open-source Hadoop implementation [34] and run two applications – (1) mapreduce-wc that performs a word count over a large corpus (5 GB), and (2) mapreduce-write that populates the file system with randomly-generated words. Performance is measured as the amount of time to perform the task. The workload involves both CPU and I/O activity.

Workload drivers: For *websearch* and *webmail*, the servers are exercised by a Perl-based client driver, which generates and dispatches requests (with user-defined think time), and reports transaction rate and QoS results. The client driver can also adapt the number of simultaneous clients according to recently observed QoS results to achieve the highest level of throughput without overloading the servers. The *ytube* workload uses a modified SPECweb2005 client driver, which has similar functionality to the other client drivers.

6.3 Metrics and Models

Representative benchmarks are only one piece in presenting a complete view of warehouse computing environments. In order to accurately evaluate these environments, it is also important to understand the key metrics specific to these environments. Furthermore, there are complex interactions within the data center that require detailed models to capture their behavior, both at a performance and total cost of ownership level.

6.3.1 Metrics

The key performance/price metric for internet sector environments is the *sustainable performance (Perf) divided by total cost of ownership (abbreviated as TCO-$).* This focus is evidenced by companies such as Google using servers that are able to provide sufficient performance at a low total cost of ownership [85], as opposed to the highest performing servers available. For the performance aspect of Perf/TCO-$, the definition specific to each workload is used (see Table 3). For total lifecycle cost, a three-year depreciation cycle is assumed and costs associated with base hardware, burdened power and cooling, and real-estate are included. In the discussion of specific trends, other metrics are also considered, such as Performance-per-Watt (Perf/W) and performance-per-infrastructure only (Perf/inf-$), and performance-per-power and cooling (Perf/P&C-$).

6.3.2 Server and Data Center Cost Models

In order to calculate the total cost of ownership, this thesis utilizes a cost model that is able to combine the costs of the main components. The two main components of the cost model are (1) the base hardware costs, and (2) the burdened power and cooling costs. For

the base hardware costs, the costs of the individual components – CPU, memory, disk, board, power and cooling (P&C) components such as power supplies, fans, etc. – are collected at a per-server level. These numbers were collected as part of a study in late 2007 [61], but their overall trends still hold true currently. These costs were cumulated at the rack level, and additional switch and enclosure costs were also considered at that level. A variety of sources were used to obtain the cost data, including publicly-available data from various vendors (Newegg, Micron, Seagate, Western Digital, etc.) and industry-proprietary cost information through personal communications with individuals at HP, Intel/AMD, ARM, etc. Wherever possible, the consistency of the overall costs and the breakdowns were also validated with prior publications from internet-sector companies [11]. For example, the total server costs were similar to that listed from Silicon Mechanics [44].

For the power and cooling costs, there are two subcomponents. The first subcomponent is the rack-level power consumption (*P_consumed*). *P_consumed* is computed as a sum of power at the CPU, memory, disk, power-and-cooling, and the rest of the board, at the per-server level, and additional switch power at the rack level. Given that nameplate power is often overrated [29], if possible the maximum operational power consumption of various components were collected from spec sheets, power calculators available from vendors [3, 26, 45, 46, 63, 73, 83], or personal communications with vendors. These power values still suffer from some inaccuracies since actual power consumption has been documented to be lower than worst-case power consumption [70]. Therefore an “activity factor” of 0.75 is used to address this discrepancy. As validation, the power model was compared to several real-world systems and was found to closely

Details	**Srvr1**	**Srvr2**
Per-server cost ($)	$3,225	$1,620
CPU	$1,700	$650
Memory	$350	$350
Disk	$275	$120
Board + mgmt	$400	$250
Power + fans	$500	$250
Switch/rack cost	$2,750	$2,750
Server power (Watt)	340	215
CPU	210	105
Memory	25	25
Disk	15	10
Board + mgmt	50	40
Power + fans	40	35
Switch/rack power	40	40
Server qty per rack	40	40
Activity factor	0.75	0.75
K1 / L1 / K2	1.33 / 0.8 / 0.667	
3-yr power & cooling	$2,457	$1,554
Total costs ($)	$5,682	$3,174

Table 4: Individual cost and power models components for two classes of servers. *The individual model components are shown for a mid-range (srvr1) and low-end (srvr2) server, along with the total costs assuming a 3-year lifespan.*

model actual consumption, usually within 15% accuracy. (A range of activity factors from 0.5 to 1.0 was also studied, and the results were qualitatively similar so they are not presented.)

Second, *P_consumed* is used as input to determine the burdened cost of power using the methodology discussed by Patel et al. [70, 71]. The equation is as follows:

$$\text{PowerCoolCost} = (1 + K_1 + L_1 + K_2 * L_1) * U_{s,\text{grid}} * P_{\text{consumed}}$$

This model assumes the burdened power and cooling costs to consist of electricity costs at the rack level, the amortized infrastructure costs for power delivery (K1), the electricity costs for cooling (L1) and the amortized capital expenditure for the cooling infrastructure (K2). For the default configuration, published data on default values were

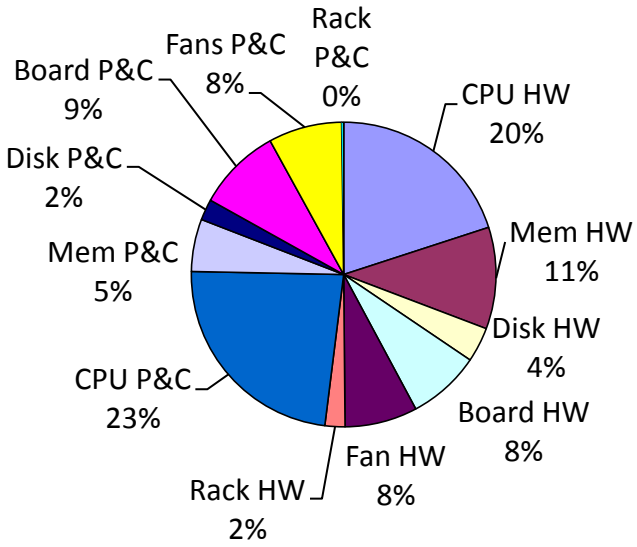


Figure 16: Contribution of cost components for s*rvr2*

A breakdown of the total contribution of each cost component to the total cost of ownership, assuming a 3 year lifespan. Note that power and cooling (P&C) costs are comparable to hardware (HW) costs.

used for K1, L1, and K2 [71]. There is a wide variation possible in the electricity tariff rate (from $50/MWhr to $170/MWhr), but this study uses a default electricity tariff rate of $100/MWhr [70]. Table 4 illustrates the breakdown of the cost model and Figure 16 shows a pie chart of the individual cost contributions (for hardware versus power and cooling) given a 3 year ownership. The two classes of servers are s*rvr1*, which represents a mid- to high-end server, and s*rvr2*, which represents a low-end server.

6.3.3 Performance Evaluation

Performance is evaluated through detailed execution-driven simulation using HP Labs' COTSon simulator [28], which is based on AMD's SimNow™ [13] infrastructure. It is a validated full-system x86/x86-64 simulator that can boot an unmodified Linux OS and execute complex applications. The simulator guest runs 64-bit Debian Linux with the 2.6.15 kernel. The benchmarks are compiled directly in the simulated machines where

applicable. Most of the benchmarks use Java, and are run using Sun's Linux Java SDK 5.0 update 12. C/C++ code was compiled with gcc 4.1.2 and g++ 4.0.4. The memory sharing is evaluated using similar trace-based simulation methodology as in Section 4.1.

6.4 A New Server Architecture

In this thesis, a detailed cost analysis is performed using the tools discussed in the previous sections, and four key areas for improvement are identified, namely CPU, packaging, memory, and disk. To address these areas, a new system architecture is designed that takes a holistic approach at improving efficiency at these four levels. The individual components of the new architecture are first discussed in isolation, and their impact is evaluated across the newly developed benchmark suite. In Section 6.5, the unification of the individual components is evaluated and shown to provide significant efficiency improvements over traditional servers.

6.4.1 Cost Analysis and Key Issues

Table 4 lists the hardware component costs, the baseline power consumption, and the burdened costs of power and cooling for two existing server configurations (*srvr1* and *srvr2*). Figure 16 presents a pie-chart breakdown of the total costs for *srvr2* separated as infrastructure (HW) and burdened power and cooling (P&C). The data shows several interesting trends. First, power and cooling costs are comparable to hardware costs. This cost trend is consistent with recent studies from internet sector workloads that highlight the same trend [29]. Furthermore, the CPU hardware and CPU power and cooling are the two largest components of total costs (contributing 20% and 23% respectively). However, it can be seen that a number of other components together contribute equally to

the overall costs. Consequently, to achieve substantial performance-per-dollar advantages, solutions need to holistically address multiple components.

This thesis designs one such holistic solution and examines it in detail. Specifically, four key issues are considered:

(1) Can overall costs from the CPU (hardware and power) be reduced by leveraging compute disaggregation and using high-volume, lower-cost, lower-power (but also lower-performance) non-server processors?

(2) Can the burdened costs of power be reduced by novel packaging solutions?

(3) Can the overall costs for memory be reduced by using disaggregated memory to share capacity across a cluster/ensemble?

(4) Can the overall costs for the disk component be reduced by using lower-power (but lower performance) disks, possibly with emerging non-volatile memory?

The goal of this chapter is to evaluate first, if considerable gains are possible in each of these areas when the architecture is viewed from the ensemble perspective rather than as a collection of individual systems, and second, if the combination of the improvements in each of these areas can lead to an overall design that improves significantly on the current state of the art server designs.

6.4.2 Compute Disaggregation

One strategy to reduce CPU costs is to leverage the principles of disaggregation and disaggregate the compute resources. The driving principle is to divide up compute resources – which are traditionally provided by high-performance, monolithic CPUs – and instead assemble the resources into smaller, more cost-effective compute units that are more properly provisioned for the workloads. Publications by large internet sector

companies such as Google [12] exhibit the usefulness of building servers using commodity desktop PC parts that are more cost-effective than server parts. The intuition is that volume drives cost; compared to servers that have a limited market and higher price margins, commodity PCs have a much larger market that allows for lower prices. Additionally, these processors do not include cost premiums for features like multiprocessor support and advanced ECC that are made redundant by reliability support in the software stack for internet sector workloads.

The unique nature of the internet sector makes a compute disaggregation-oriented approach feasible. Whereas servers for databases or HPC have traditionally focused on obtaining the highest performance-per-server, the scale-out nature of the internet sector allows for a focus on performance-per-dollar by utilizing systems that offer a superior efficiency. Due to the massive scale of the internet sector, applications have been geared towards a scale-out infrastructure that allows extra performance to be obtained by adding more servers. In addition, these infrastructures are designed to be tolerant to the faults that are encountered at high-scale, such as computer or network failure. Therefore it is possible to leverage this scale-out orientation to support using smaller, more efficient servers in a manner that offers lower-cost and comparable performance to traditional, monolithic servers.

This section *quantitatively* evaluates the benefits of varying degrees of compute disaggregation, studying the effectiveness of low-end servers and desktops for warehouse-computing environments. This focus on performance-per-dollar is taken one step further by exploring an alternative commodity market – the embedded/mobile segment. Trends in transistor scaling and embedded processor design have brought

System	"Similar to"	Processors x Cores	Speed	Arch.	L1/L2 cache size	Watt	Inf-$
Srvr1	Intel Xeon MP, AMD Opteron MP	2p x 4 cores	2.6 GHz	OoO	64K/8MB	340	3,294
Srvr2	Intel Xeon, AMD Opteron	1p x 4 cores	2.6 GHz	OoO	64K/8MB	215	1,689
Desk	Intel Core 2, AMD Athlon 64	1p x 2 cores	2.2 GHz	OoO	32K/2MB	135	849
Mobl	Intel Core 2 Mobile, AMD Turion	1p x 2 cores	2.0 GHz	OoO	32K/2MB	78	989
Emb1	AMD Embedded Athlon 64	1p x 2 cores	1.2 GHz	OoO	32K/1MB	52	499
Emb2	Intel Atom, ARM Cortex-A5	1p x 1 core	600 MHz	Inorder	32K/128K	35	379

Table 5: Summary of systems considered.

Multiple classes of systems are considered in this study, ranging from a mid-range server (srvr1) to a low-end embedded class system (emb2). The processor architecture type (Arch) is also listed; the options are inorder or out-of-order (OoO).

powerful, general purpose processors to the embedded space, many of which are multicore processors. The on-going development of Intel's Atom platform [47] and Qualcomm/ARM's Snapdragon platform [74] is indicative of the powerful and versatile embedded-class processors that are currently available. Devices using embedded CPUs are shipped in even more volume than desktop systems, leading to even higher cost savings. Additionally, they are often designed for minimal power consumption due to their use in mobile systems. As shown in Section 6.4.1, power is a large portion of the total lifecycle costs, so greater power-efficiency can help reduce costs. The key open question is whether these cost and power benefits can offset the performance degradation relative to the baseline server.

Six different system configurations are considered in this study and are detailed in Table 5. *Srvr1* and *srvr2* represent mid-range and low-end server systems; *desk* represents desktop systems, *mobl* represents mobile systems, and *emb1* and *emb2*

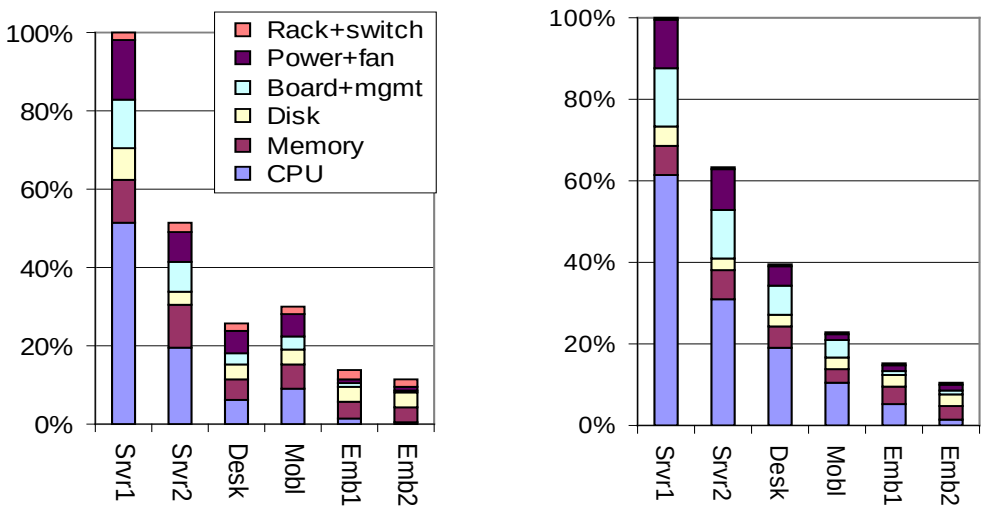


Figure 17: Infrastructure and power and cooling cost comparisons across multiple system classes.

(a) Infrastructure cost breakdown for the six system classes. (b) Power and cooling cost breakdown for the system classes. Note the significant decrease in cost as commodity parts are used.

represent a mid-range and low-end embedded system respectively. All servers have 4 GB of memory, using FB-DIMM (*srvr1, srvr2*), DDR2 (*desk, mobl, emb1*) or DDR1 (*emb2*) technologies. *Srvr1* has a 15,000 RPM disk and a 10 Gigabit NIC, while all others have a 7,200 RPM disk and a 1 Gigabit NIC. Note that the lower-end systems are not balanced from a memory provisioning standpoint (reflected in the higher costs and power for the lower-end systems than one would intuitively expect). However, the goal of this study is to isolate the effect of the processor type, so memory and disk capacity are kept constant (but in the different technologies specific to the platform). Later sections examine changing this assumption.

Evaluation: Figure 17 and Table 6 present the evaluation results. The breakdown of infrastructure costs and the burdened power and cooling costs are summarized in Figure 17(a) and Figure 17(b) respectively. Table 6 shows the variation in performance,

	Workload	Srvr2	Desk	Mobl	Emb1	Emb2
Perf	websearch	68%	36%	34%	24%	11%
	webmail	48%	19%	17%	11%	5%
	ytube	97%	92%	95%	86%	24%
	mapred-wc	93%	78%	72%	51%	12%
	mapred-wr	72%	70%	54%	48%	16%
	HMean	71%	42%	38%	27%	10%
Perf/Inf-$	websearch	133%	139%	112%	175%	93%
	webmail	95%	72%	55%	83%	44%
	ytube	188%	358%	315%	629%	206%
	mapred-wc	181%	302%	241%	376%	101%
	mapred-wr	141%	272%	179%	350%	140%
	HMean	139%	162%	125%	201%	91%
Perf/W	websearch	107%	90%	147%	157%	103%
	webmail	76%	47%	73%	75%	49%
	ytube	152%	233%	413%	566%	229%
	mapred-wc	146%	197%	315%	338%	113%
	mapred-wr	114%	177%	235%	315%	157%
	HMean	112%	105%	164%	181%	101%
Perf/TCO-$	websearch	120%	113%	124%	167%	97%
	webmail	86%	59%	62%	80%	46%
	ytube	171%	291%	351%	600%	215%
	mapred-wc	164%	246%	268%	359%	106%
	mapred-wr	128%	221%	200%	334%	147%
	HMean	126%	132%	140%	192%	95%

Table 6: Summary of benefits from using low-cost low-power CPUs from non-server markets.

Performance, cost, and power efficiencies of the different systems are shown, relative to a srvr1 baseline.

performance/$ and performance/Watt. For better illustration of the benefits, performance/$ is shown as performance/total costs and performance/infrastructure costs (performance/power-and-cooling-costs can be inferred). Also listed is the average computed as the harmonic mean of the throughput and reciprocal of execution times across the benchmarks.

Figure 17(a), demonstrates that, at a per-system level, the hardware costs are dramatically reduced for the consumer systems. The biggest costs reductions come in the CPU component; the use of consumer technologies, like DDR2 for memory, lead to

reductions in other components as well. The *desk* system is only 25% of the costs of the *srvr1* system, while the *emb1* is only 15% of the costs. The *mobl* system sees higher costs relative to the desktop because of the higher premium for low-power components in this market. Similar trends can be seen for power and cooling costs in Figure 17(b). As one would expect, the *desk* system has 60% lower P&C costs compared to *srvr1*, but the *emb1* system does even better, saving 85% of the costs. Unlike hardware costs, there is a more gradual progression of savings in the power and cooling.

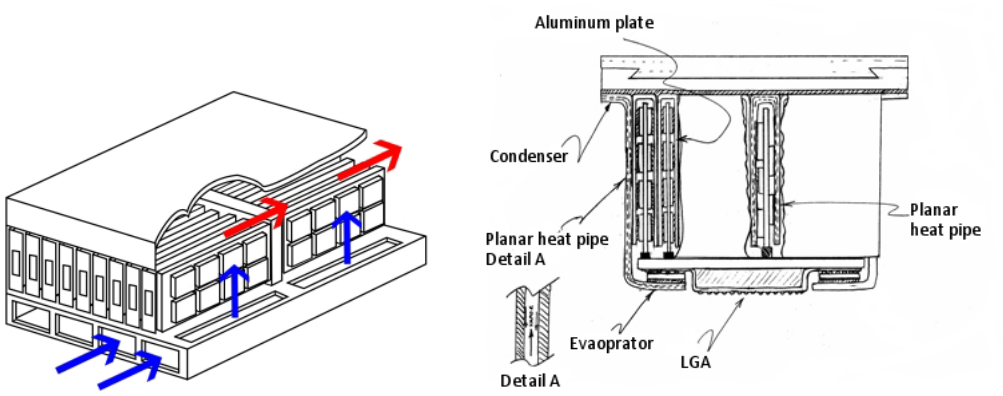
Table 6 highlights several interesting trends for performance. As expected, the lower-end systems see performance degradation compared to *srvr1*. However, the relative rate of performance degradation varies with benchmark and the system considered. The *mapreduce* workloads and *ytube* see relatively smaller degradations in performance compared to *websearch* and *webmail* and also see a much more dramatic inflection at the transition between *emb1* and *emb2* systems. These performance changes are intuitive given that these workloads are not CPU-intensive and are primarily network or disk bound. The desktop system sees 10-30% performance degradation for *mapreduce* and *ytube* and 65-80% performance loss for *websearch* and *webmail*. In comparison the *emb1* system sees 20-50% degradations for the former two workloads and 75-90% loss for the remaining two. *Emb2* consistently underperforms for all workloads.

Comparing the relative losses in performance to the benefits in costs, there are significant improvements in performance/Watt and performance/$ for the *desk*, *mobl*, and *emb1*. On the other hand, *emb2* does not perform as well. For example, *emb1* achieves improvements of 300% to 600% higher performance/TCO-$ for *ytube* and *mapreduce* and an improvement of 60% for *websearch* compared to *srvr1*. *Webmail* achieves a net

degradation in performance/TCO-$ because of the significant decrease in performance, but *emb1* still performs competitively with *srvr1*, and does better than the other systems. Performance/Watt results show similar trends except for stronger improvements for the mobile systems.

Overall, the workloads show improved performance per costs when using lower-end consumer platforms optimized for power and costs, compared to servers such as *srvr1* and *srvr2*. The *desk* configuration performs better than *srvr1* and *srvr2* validating published practices of using commodity desktops computers [12]. However, a key new interesting result for this benchmark study is that leveraging compute disaggregation and using embedded systems has the potential to offer more cost savings at the same performance. The choice of embedded platform is important – *emb2* significantly underperforms compared to all other configurations due to its significantly lower processing speed. These results do not necessarily preclude such processors from being effective in the internet sector space, but do indicate that significant architectural changes would be required. It must be noted that these results hold true for the benchmark suite developed in this work, but more study is needed before these results can be generalized to other variations of internet sector workloads.

Studying embedded platforms with larger amounts of memory and disk added non-commodity costs to the model that can be further optimized through use of designs such as memory disaggregation. Additionally, *srvr1* consumes 13.6KW/rack while *emb1* consumes only 2.7KW/rack (for a standard 42U rack). This rack-level power consumption difference is not factored into results, but this difference can either translate



(a) (b)

Figure 18: New proposed cooling architectures.
(a) Dual-entry enclosure with directed airflow. (b) Aggregated cooling of microblades.

into simpler cooling solutions, or smaller form-factors and greater compaction. The next section addresses the latter.

6.4.3 Compaction and Aggregated Cooling

After the processor, inefficiencies in the cooling system are the next largest factor in total cost. Lower-power systems offer the opportunity for smaller form factor boards, which in turn allow for optimizations to the cooling system. This section discusses two such optimizations, using blade servers as the base architecture for the rest of the discussions because they are well known and widespread in the market. This work was done in close collaboration with experts on thermal-management techniques and research from Hewlett-Packard Labs, and utilizes previously patented techniques [69] to enable new optimizations.

Dual-entry enclosures with directed airflow: Figure 18(a) shows how a server level enclosure can be redesigned to enable blades to be inserted from front and back to attach to a midplane. The key intuition is to partition the air flow, and allow cold air to be

directed vertically through the blades. This direction of air flow is done by increasing the volume of the enclosure to create an inlet and exhaust plenum, and direct the air flow in the directions indicated by the arrows in the picture. The air flow is maintained through all the blades in parallel from intake plenum to exhaust plenum. (This configuration is akin to a parallel connection of resistances versus a serial one.) Compared to conventional blade enclosures which force air directly from front to back, this air flow configuration results in shorter flow length (distance traversed by the air), lower pre-heat (temperature of the air hitting the blades), reduced pressure drop and volume flow. Thermo-mechanical modeling software was to analyze the thermal resistance air flow improvements with this design, and it shows significant improvements in cooling efficiencies (~50%). Compared to the standard baseline that has 42 1U “pizza box” servers per rack, the new design can allow 40 blades of 75W to be inserted in a 5U enclosure, allowing 320 systems per rack.

Board-level aggregated heat removal: Figure 18 also shows an even more radical packaging design. With low-power systems, one can consider blades of much smaller form factors that are integrated on conventional blades that fit into an enclosure. As illustrated in Figure 18(b), an innovative packaging scheme is proposed that aggregates the power dissipating components at the device and package level. The smaller form factor server modules are interspersed with planar heat pipes that transfer the heat at an effective conductivity three times that of copper to a central location. The aggregated heat is removed with a larger optimized heat sink that enables channeling the flow through a single heat sink as opposed to multiple separate conventional heat sinks. The increased conductivity and the increased area for heat extraction lead to more effective cooling. The

smaller blades can be connected to the blades through different interfaces – TCA or COMX interfaces are good candidates [72]. Figure 18(a) shows an example with eight such smaller 25W modules aggregated on a bigger blade. With higher power budgets, four such small modules can be supported on a bigger blade, allowing 1250 systems per rack.

These cooling optimizations have the potential to achieve two to four times greater efficiencies than conventional server cooling designs. Additionally, although they use specialized designs, these cooling solutions should be effective in other enterprise environments. When combined with the significant and growing fraction of the market represented by warehouse computing environments, these designs should have enough volume to drive commoditization.

6.4.4 Memory Disaggregation

Memory costs and power are an important part of the system level picture, especially as the cost and power of other components are reduced. Yet at a datacenter level, it can be difficult to properly choose the amount of memory in each server. As discussed in previous sections, the memory demands across workloads vary widely, and past studies have shown that per-server sizing for peak loads can lead to significant ensemble-level overprovisioning [29, 76]. To address memory overprovisioning, this work examines using the previously proposed memory disaggregation as a component of the new server architectures discussed in this chapter. Using the memory blade, memory is provisioned at a coarser granularity (i.e., per blade enclosure), sizing each larger unit to meet the expected aggregate peak demand. This allows the attached servers to have smaller local memories by exploiting locality to maintain high performance; these reduced memories

thereby aid in compaction. By allowing memory to be right provisioned, disaggregated memory achieves power and costs savings, and enables further server compaction.

Memory disaggregation design for the internet sector: Based on the performance results in Chapter 4, the PS design is assumed for disaggregated memory. This design connects the memory blade and compute blades using a commodity interconnect, assumed to be a PCIe 2.0 x2 link. The PS design focuses on ensuring the modifications to enable access to the memory blade are only made at the software level. Specifically, the hypervisor is modified to detect access to remote memory at a page granularity, and it swaps remote pages into local memory upon access. Although this design achieves high performance by leveraging locality, the remote-to-local page-granularity transfers incur relatively high latencies (approximately 4-5 µs) because an entire 4 KB page must be transferred prior to use. Additionally, as discussed in Chapter 5, based on the remote memory implementation and the processor's architecture there may be additional latency from detecting the remote access. These transfer latencies can be one of the primary factors in overall performance of disaggregated memory.

The previously proposed disaggregated memory architecture designs did not consider any aggressive optimizations to address the transfer latencies because they were focused on simplicity and minimal changes. Optimizations such as modification of the CPU architecture were avoided to minimize non-commodity component costs. However, the large scale of internet sector data centers can make domain-specific architecture designs economically feasible, and can additionally drive commoditization of those architectures. Thus, in addition to evaluating the baseline PS architecture on the newly developed workload suite, this section also considers a hardware-based optimization to reduce

	websearch	webmail	ytube	mapred-wc	mapred-wr
PCIe x2 (4 µs)	4.7%	0.2%	1.4%	0.7%	0.7%
CBF (0.5 µs)	1.2%	0.1%	0.4%	0.2%	0.2%

(a)

	Perf/Inf-$	Perf/Watt	Perf/TCO-$
Static	102%	116%	108%
Dynamic	106%	116%	111%

(b)

Table 7: Memory sharing architecture and results.

(a) Slowdowns using random replacement for 25% first-level memory size. (b) Net cost and power efficiencies given different remote memory partitioning schemes.

transfer latencies as a future-looking option. Specifically, modest support in the local memory controller is used to implement a critical-block-first (CBF) optimization. This additional logic would use the coherence protocol to stall accesses to a block on a remote page in transit until the block is transferred. This scheme allows the address mapping for a remote page to be installed as soon as the incoming page transfer is set up, thereby allowing the faulting access to be completed as soon as the needed block (which would be scheduled first) arrives. If the other blocks of the remote page are not accessed prior to the completion of the transfer, then the transfer latency can be almost completely hidden beyond the latency for the first block. Based on PCIe access latency and bandwidth, and packet creation latencies, a total latency of 0.75 µs is assumed for retrieving the first block of a page.

Performance evaluation: As in Section 4.1, trace-driven simulation is used for performance evaluation. Traces on the benchmark suite are gathered using the *emb1* processor model, which is the deployment target in Section 6.4.2. Using the *emb1* model ensures that the memory access traces match the accesses generated by that microarchitecture. The trace simulator is used to model varying local memory sizes, and

transfer latencies of 4 µs for 4KB pages on a PCIe 2.0 x2 link, and 0.75µs for the critical-block-first (CBF) optimization are assumed.

Using the baseline of *emb1*, Table 7(a) shows the relative performance of the two-level configuration using random replacement. Baselines of both 4 GB and 2 GB are studied, but only the 2 GB results are reported here to be conservative about the scaled down workloads. Two scenarios are considered, one where 25% of the server's memory is local, and the other where 12.5% of the memory is local. These scenarios assume aggressive reduction of local memory to enable greater form-factor compaction. The results show that baseline disaggregated memory can cause slowdowns of up to 5% and 10% for the 25% and 12.5% local-remote split, respectively. By including the CBF optimization, these slowdowns are reduced to ~1% and 2.5% for the 25% and 12.5% local-remote split, respectively. The workloads with larger memory usage, *websearch* and *ytube*, have the largest slowdown. Interestingly, the CBF optimization has a greater impact than increasing the local memory capacity, indicating its potential in enabling aggressive compaction. These initial results indicate that a two-level memory hierarchy with a first-level memory of 25% of the baseline would likely have minimal performance impact even on larger workloads.

Cost evaluation: The cost and power savings are calculated for two scenarios, both assuming that the processor blades have 25% of the baseline's local memory, while remote memory blades use DRAM devices that are in the "sweet spot" of the commodity market, being slower but 24% cheaper [26]. The same methodology is used as in Section 4.1, but with slightly different technology parameters to match the other components of this study. The *static partitioning* scheme uses the same total amount of DRAM as the

baseline, with the remaining capacity (75%) on the memory blades. This scheme assumes a straightforward design that does not assume the extra software support needed to enable dynamic provisioning. The *dynamic provisioning* scheme assumes that 20% of the blades use only their local memory, allowing the total system memory to be 85% of the baseline (25% local, 60% on memory blades). This scheme models scenarios where data center memory usage is expected to see large variations, and thus benefit from dynamic provisioning. Based on the results in Table 7(a), a 2% slowdown is assumed across all benchmarks. The cost evaluation results, shown in Table 7(b), demonstrate that both schemes provide good benefits (16%) in system-level performance/P&C-$. The static scheme provides a negligible (2%) improvement system performance/Inf-$, since it uses the same total amount of DRAM, but the dynamic scheme gains an additional 6% improvement by reducing the total necessary DRAM. These figures combine to give 8% (static) and 11% (dynamic) benefits in performance/TCO-$.

6.4.5 Flash as Disk-cache with Low-power Disks

Continuing the theme of using lower-power components and optimizing for the ensemble, this section addresses the benefits of using lower-power laptop disks. In addition to lower power, laptop disks have the benefit of a smaller form factor allowing greater compaction for aggregated cooling (such as the designs in Section 6.4.3). However, these improvements are offset by lower performance and a higher price. These experiments consider using the laptop disks moved to a basic Storage Area Network (SAN) connected through the SATA interface. By utilizing a SAN, individual server blades do not have to be physically sized to fit a disk, allowing the small module form factor in Section 6.4.3 to be used.

	Flash	Laptop Disk	Laptop-2 Disk	Desktop Disk
Bandwidth	50 MB/s	20 MB/s	20 MB/s	70 MB/s
Access time	20 µsec read 200 µsec write 1.2 msec erase	15 msec avg. (remote)	15 msec avg. (remote)	4 msec avg. (local)
Capacity	1 GB	200 GB	200 GB	500 GB
Power (W)	0.5	2	2	10
Price	$14	$80	$40	$120

(a)

Disk Type	Perf/Inf-$	Perf/Watt	Perf/TCO-$
Remote Laptop	93%	100%	96%
Remote Laptop + Flash	99%	109%	104%
Remote Laptop-2 + Flash	110%	109%	110%

(b)

Table 8: Low-power disks with flash disk caches.

(a) Listing of key flash and disk parameters. Note that the laptop disks assume a remote SAN configuration. (b) Net cost and power efficiencies of using laptop disks, and flash-based disk caching.

Additionally, these experiments also examine the use of non-volatile flash technology. As shown in Table 8(a), Flash has desirable power, performance, and cost characteristics well aligned with the goals of the overall architecture. However, one of the limitations of using Flash is that it "wears out" after approximately 100,000 writes (assuming current technology). While this limitation is an important drawback, predicted future technology improvements and wear-out leveling schemes [50] can greatly increase Flash memory life time. When these advances are considered along with typical 3-year depreciation cycles and software-failure tolerance in internet workloads, it is likely that Flash memory can feasibly be used in internet sectors. These experiments specifically explore the use of a flash-based disk caching mechanism [50], with the flash being located on the server board itself. The flash caches any pages that were recently accessed from disk. The flash cache augments the OS's page cache which is maintained in DRAM. Any time a page is

not found in the OS's page cache, the flash cache is searched by looking up in a software hash table to see if the flash holds the desired page.

Evaluation: The experiments in this section evaluate the achieved efficiency by using a remote SATA laptop drive (with very conservative bandwidth and latency values to account for possible SAN performance penalties) with the *emb1* configuration. Performance numbers are obtained with and without a 1 GB flash cache located on the server board. The runs are normalized to the baseline configuration of having a local desktop-class disk, and the configurations are listed in Table 8(a). Note that in favor of simplicity, a single-flash device configuration is used; higher bandwidths are possible by interleaving accesses across multiple flash devices. Also note that the laptop class disks are remote to the server, while the desktop class disk is local, accounting for the latency difference. The two laptop disks have identical performance, but Laptop-2 assumes that prices are driven down by commoditization.

The results in Table 8(b) show that using low-power laptop disks alone is *not beneficial* from a performance/$ perspective for the benchmarks. The loss in performance dominates the savings in power. However, using a flash-based disk cache provides an 8% performance improvement over the remote laptop disk, providing better performance/$ compared to the baseline desktop case. The achieved efficiencies with a cheaper laptop disk (laptop-2) show better results (close to 10% better performance/$), pointing to greater benefits if laptop disk prices decline to desktop disk levels. This may be a realistic scenario as laptop drives become more commoditized.

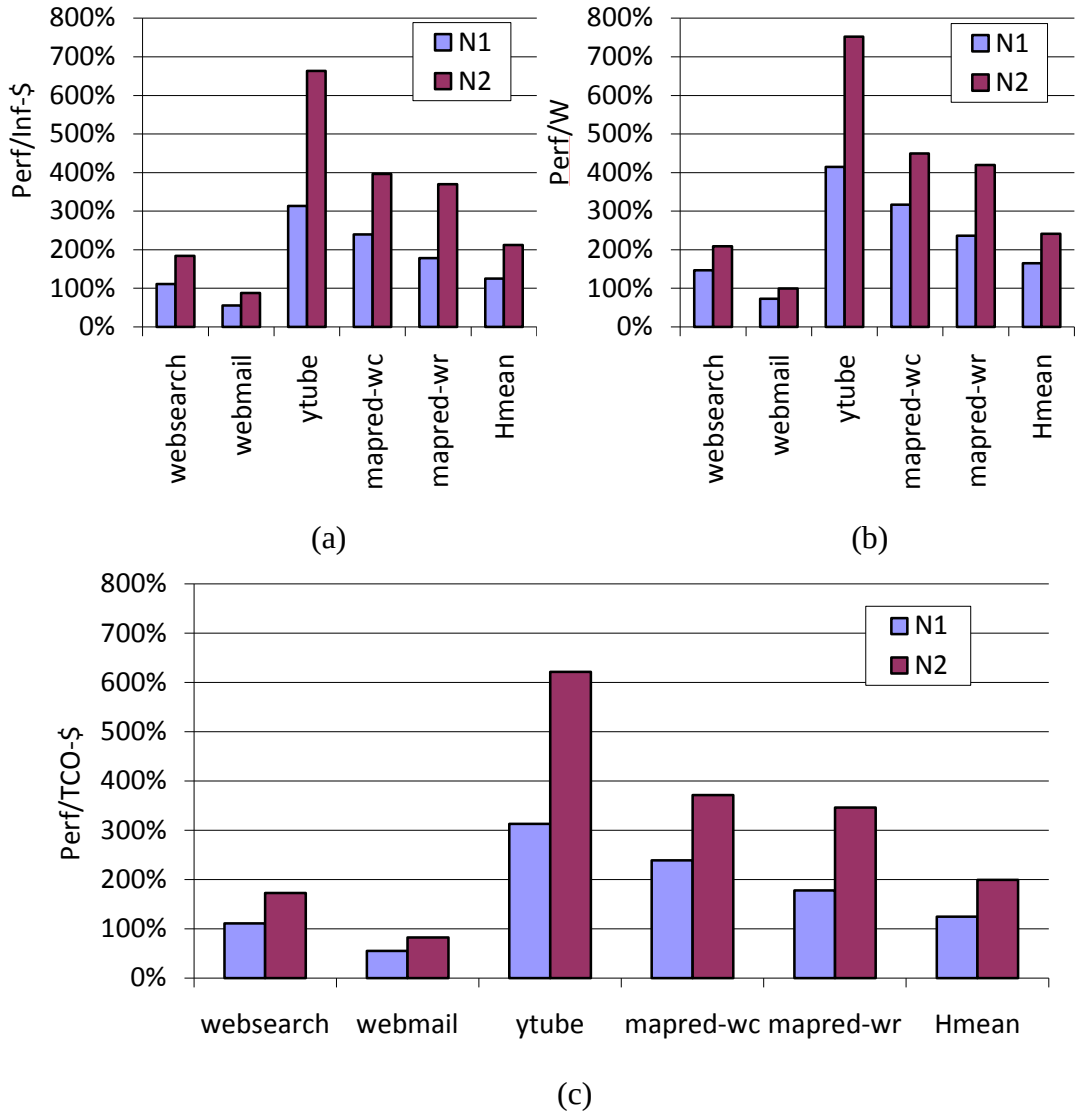


Figure 19: Cost and power efficiencies for the two unified designs.

Efficiencies achieved relative to a baseline srvr1 system. (a) Performance-per-Infrastructure cost efficiencies. (b) Performance-per-Watt efficiencies. (c) The key result of Performance-per-Total Cost of Ownership efficiencies is shown.

6.5 Unifying the Architectural Solutions

So far, this thesis has described solutions targeted at specific subsystems and evaluated their benefits in isolation. In this section, these optimizations are shown to work together and their combined benefits are evaluated.

6.5.1 Two Unified Designs

Based on the observations in Section 6.1, two new server architectures are considered for the internet sector. The *N1* design represents a solution practical in the near-term; it uses mobile-class blades with dual-entry enclosures and directed airflow, but does not include disaggregated memory or flash-based disk caching with mobile disks. The *N2* design represents a likely longer-term solution; it uses embedded blades with aggregated cooling housed in an enclosure with directed air-flow. Unlike *N1*, the *N2* design uses disaggregated memory and remote low-power disks with flash-based disk caching, which enables its level of compaction. Some of the changes required for the *N2* configuration assume custom components, but as discussed, the changes are likely to become cost-effective in a few years with the volumes in this market.

6.5.2 Evaluation

Figure 19 shows how the two solutions provide significant improvements to cost and power efficiencies compared to the baseline *srvr1* system. Focusing on the *ytube* and *mapreduce* benchmarks, the performance/TCO-$ (Figure 19(c)) improves by 100% to 250% for the current-generation solution (*N1*) and by 250% to 500% for the next-generation solution (*N2*). Figure 19(a) and Figure 19(b) show that these benefits are equally from savings in both infrastructure costs and power. As before, *websearch* gets lower benefits of 10%-70% improvement, and *webmail* sees degradations (40% for *N1*, and 20% for *N2*). Figure 19(c) also shows the harmonic mean across the benchmarks. Overall, the two new solutions can improve sustained throughput per total infrastructure dollar by 50% (with *N1*) to 100% (with *N2*). The same result can be restated differently. For the same performance as the baseline, *N2* gets a 60% reduction in power, and 55%

reduction in overall costs, and consumes 30% less racks (assuming 4 embedded blades per blade, but air-cooled).

Though not presented here, the new proposed solutions were also compared to a baseline based on *srvr2* and *desk*, and it is found that there continues to be significant benefits. *N2*, in particular, gets average improvements of 80% to 100% over the corresponding baselines, and the *ytube* and *mapreduce* benchmarks get 150% to 310% and 70% to 150% better performance/$ compared to *srvr2* and *desk*, respectively. *N1* also continues to get better performance for *ytube* and *mapreduce*, but the benefits are also scaled down (10% to 100%).

6.6 Discussion

While this work is a first step in the study and optimization of warehouse-computing environments, several caveats and opportunities for future work exist.

6.6.1 Benchmark Suite

The best efforts were made to ensure the benchmarks are as realistic as possible, using real-world traces and insights from previous studies. It was additionally ensured that the benchmarks replicate behavior described in the limited public documentation in this space. Despite this strive for accuracy, there are certain limitations in the current suite. In actual deployments, requests follow a time-of-day distribution [29], but this study only uses request distributions that focus on sustained performance. Real deployments operate on much larger data sets, but often data sizes had to be scaled for practical simulation times while striving to keep the same trends (e.g., the scaled *websearch* workload matches prior published work with larger-scale experiments [52]). Also, only one

representative benchmark that best mimics previously-published data was studied for each workload. In reality, a workload (e.g., *websearch*) can have multiple flavors based on the nature of the request (e.g., user queries, data set, etc). Further study is needed to understand the applicability of the results of the benchmark suite to all internet sector workloads. Given different workload behavior (e.g., heavily CPU bound and I/O intensive), it is possible that *srvr1* will be a more effective solution than *emb1*, given its higher compute power and I/O bandwidth.

6.6.2 Metrics and Models

The design of good models that fold single-system metrics into a more complex cluster level metric is an ongoing area of research. In the absence of such models and practical cluster-level full-system simulation, the performance models used make the simplifying assumption that cluster-level performance can be approximated by the aggregation of single-machine benchmarks. This assumption needs to be validated, by prototyping, or by study of real environments.

The cost model data was largely collected from a variety of public sources. However, a few of the values were derived from confidential communication and industry estimates, and more work is needed in deriving an open and public cost model for the broader community. Ideally, personnel and real-estate costs, though harder to characterize, would also be included in such a model.

6.6.3 Amdahl's Law Limits on Scale-out

The newly proposed solutions assume that the workload can be partitioned to match the new levels of scale-out. However, this scale-out cannot be taken to extremes. Particularly, for workloads like web search, there are potential overheads with this

approach in terms of decreased efficiency of software algorithms, increased sizes of software data structures, increased latency variability, greater networking overheads, etc. The minimum capacity and balance at the individual server where Amdahl's law factors in is again an interesting open question for future studies. This is an important caveat since it can bias the conclusions towards overestimating benefits for smaller platforms for some workloads. Similarly, this study only addresses latency to the extent of the individual benchmark definitions have QoS constraints on query response times (*websearch, webmail*). More aggressive simplifications of the platform can have implications on single-system performance.

6.7 Summary

Emerging internet sector companies using "warehouse scale" computing systems represent a large growth market. Their large volume and willingness to try custom solutions offer an interesting and unique opportunity to consider new server architectures with a strong emphasis on cost, power and cooling, and scale-out. Because of the novelty of these environments, this work has sought to develop benchmarks and metrics to better understand this space, and to leverage this understanding to design solutions that can achieve significant improvements in efficiency.

This thesis makes several important contributions: a new benchmark suite is assembled that is intended to model workloads and behavior common to the internet sector. Cost models and evaluation metrics are developed that are specific to this space, including an overall metric of performance per unit total cost of ownership (Perf/TCO-$). There are four key areas identified for improvement from the Perf/TCO-$ perspective (CPU, packaging, memory, and disk), and initial solutions are studied that provide benefits

relative to the status quo in each of these areas. This work shows that compute disaggregation – through the use of smaller, embedded-class processors instead of server-class processors – in complement with memory disaggregation can provide significant Perf/TCO-$ advantages for the benchmarks. This thesis also proposes and evaluates novel ensemble-level Perf/TCO-$ optimizations in packaging and cooling, and uses it in conjunction with the flash-based disk caching. Simulation results show that these techniques are beneficial individually, and more importantly together, they demonstrate the potential for significant improvements: 2 times better performance/TCO-$ on average, and 3.5 to 6 times improvement on many of the benchmarks.

Chapter 7

Related Work

There is a significant amount of related work on the individual principles of the disaggregated memory concept. There has been research into utilizing remote memory capacities, sharing memory capacities, increasing memory capacity, and multi-level main memory hierarchies. However, these related works often consider each topic in isolation, and often do not focus on the importance of commodity based designs. The work in this thesis leverages the related works to identify opportunities to redesign the memory subsystem in a way that successfully achieves the multiple goals for optimizing memory capacity.

Unlike memory subsystems, warehouse computing environments are a relatively new area. The work done in this thesis is novel in providing insight into these environments, modeling their workloads, and evaluating new system architectures targeted to that sector. There have been recent works that follow similar principles as presented in Chapter 6. These works provide qualitative validation for the results and approach taken in this thesis.

7.1 Disaggregated Memory

7.1.1 Remote Memory Capacity

There has been previous work to this thesis that has also examined using remote memory capacity. A large body of prior work (e.g., [31, 30, 33, 22, 41, 43, 53]) has examined using remote servers' memory for swap space [31, 41], file system caching [30, 22], or RamDisks [33], typically over conventional network interfaces (i.e., Ethernet). These approaches do not fundamentally address the compute-to-memory capacity imbalance: the total memory capacity relative to compute is unchanged when all the servers need maximum capacity at the same time. Additionally, although these approaches can be used to provide sharing, they suffer from significant limitations when targeting commodity-based systems. In particular, these proposals may require substantial system modifications, such as application-specific programming interfaces [53] and protocols [33, 43]; changes to the host operating system and device drivers [31, 30, 33, 41]; reduced reliability in the face of remote server crashes [30, 41]; and/or impractical access latencies [33, 43].

7.1.2 Shared Memory Capacity

Much of the previous work on shared memory capacity among multiple processors, either through hardware or software, focuses on providing a large, global address space so that several processors can work on a task in parallel, and communicate through memory. Although this configuration is very important for certain market segments, for the data centers and commodity servers targeted in this thesis, such a shared configuration is not beneficial because of the distributed nature of their target workloads.

Thus providing a large, shared address space is not the end goal of disaggregated memory. By scaling back the scope and focusing solely on sharing capacity – as opposed to providing a single, unified large capacity – significant implementation complexities are avoided. Specifically the memory blade does not need to take part in the coherency domain of multiple blades, and thus does not need the interconnect and processing capabilities to handle cache-coherent communication from multiple sources.

Some examples of designs to provide a global address space include symmetric multiprocessors (SMPs) and distributed shared memory systems (DSMs) [2, 59, 55, 16, 60, 81, 5, 82, 39], which allow all the nodes in a system to share a unified address space. However, like the approaches discussed in the previous section, these designs do not target the compute-to-memory-capacity ratio. Hardware shared-memory systems typically require specialized interconnects and non-commodity components that add costs; in addition, signaling, electrical, and design complexity increase rapidly with system size. Software DSMs [60, 81, 5, 82] can avoid these costs by managing the operations to send, receive, and maintain coherence in software, but come with practical limitations to functionality, generality, software transparency, total costs, and performance [37]. A recent commercial design in this space, Versatile SMP [80], uses a virtualization layer to chain together commodity x86 servers to provide the image of a single larger system, but the current design requires specialized motherboards, I/O devices, and non-commodity networking, and there is limited documentation on performance benefits, particularly with respect to software DSMs.

7.1.3 Increasing Memory Capacity

To increase the compute-to-memory ratio directly, researchers have proposed compressing memory contents [25, 27] or augmenting/replacing conventional DRAM with alternative devices or interfaces. Recent startups like Virident [94] and Texas Memory [89] propose the use of solid-state storage, such as NAND Flash, to improve memory density albeit with higher access latencies than conventional DRAM. From a technology perspective, fully-buffered DIMMs [36] have the potential to increase memory capacity but with significant trade-offs in power consumption, as discussed in Chapter 2. There are designs that attempt to map larger amounts of DRAM onto conventional DIMM slots, such as Cisco's Extended Memory Blade server [20] or the now-defunct MetaRAM. 3D die-stacking [51] allows DRAM to be placed on-chip as different layers of silicon; in addition to the open architectural issues on how to organize 3D-stacked main memory, this approach further constrains the extensibility of memory capacity. In contrast to the work in this thesis, none of these approaches enable memory capacity sharing across nodes. In addition, many of these alternatives provide only a one-time improvement, thus delaying but failing to fundamentally address the memory capacity wall. Many of these designs force significant changes on commodity designs, limiting their usefulness at a larger scale. Furthermore, the disaggregated memory architectures in this thesis are largely orthogonal to these increased memory capacity designs and would be able to utilize them if there was a significant benefit.

Phase change memory (PCM) [57] is emerging as a promising alternative to increase memory density. However, current PCM devices suffer from several drawbacks, such as

high energy requirements, slow write latencies, and finite endurance, that limit their straightforward use as a main memory replacement.

7.1.4 Multi-level Memory Hierarchies

A recent study [27] demonstrates the viability of a two-level main memory organization that can tolerate increased access latency due to compression, heterogeneity, or network access to second-level memory. However, that study does not discuss a commodity implementation for x86 architectures or evaluate sharing across systems.

Another recent study examined using a virtual machine monitor to prototype a hybrid main memory consisting of a first level of DRAM, and a second level of Flash [98]. However work was done on a proprietary hypervisor whose source code is not readily available to the public. The work done on the hypervisor-based disaggregated memory prototype in Chapter 5 is closely related to this work, with slightly different end goals. Whereas their work seeks to model different types of memory technologies for the second level of memory, the work in this thesis seeks to model different policies for multiple systems sharing a remote memory capacity.

7.2 Extending Disaggregation

7.2.1 Warehouse Computing Workloads

Much of the related work regarding the benchmark suite has been published subsequent to when the studies in this thesis were completed. One of the prior works, which was the inspiration for parts of the benchmark suite described in Section 6.2, was by Fan et al. In that study, the authors discuss webmail, websearch, and mapreduce as three workloads used in the Google datacenter [29], but do not provide benchmarks for

those workloads. The benchmark suite in this thesis addresses these three workloads and adds an additional workload to model Youtube™-like behavior. Other benchmarks modeling web 2.0 infrastructures include a proposal from Sun called Web20Bench [88], which focuses on one interactive benchmark. Subsequent to the work in this thesis, there has been additional benchmark suites developed such as Berkeley's CloudStone, which seeks to evaluate web 2.0 applications running on cloud infrastructures using an interactive social-calendar workload [87].

7.2.2 Compute Disaggregation

There are very few contemporary works that look at embedded processors in the space of the internet sector. Concurrently with the studies performed in this thesis, work by James Hamilton on "Microslice" servers reached a similar conclusion regarding the effectiveness of low-power, low-cost processors for this class of workloads [40]. A later study by Reddi et al. at Microsoft [77] also reached similar conclusions specifically for a large scale web search application. This study also discusses some of the infrastructure-level issues that occur as a result of using lower performing processors, such as greater performance variation.

Subsequently, works by Andersen et al. [6], Caulfield et al. [19], and Szalay et al. [90] have followed similar principles and looked at even more aggressive designs with processors similar to the *emb2* class (the lowest-end embedded processor) to address specific subsets of workloads such as web servers and data-intensive applications. Although the findings in Section 6.4.2 indicate that an *emb2* class processor is not able to achieve performance gains, the architectures in these works are able to be successful by being carefully designed for and targeted towards data-intensive applications. This focus

puts them in a slightly different market segment than the server architectures proposed in this thesis, and limits their applicability to general internet sector workloads.

7.2.3 Flash in the Disk Subsystem

Previously NOR Flash was proposed as a replacement or supplement to disk [96]. The use of NAND flash as a disk buffer was proposed by Kgil and Mudge [50]; this thesis uses this approach with a similar methodology but aimed at internet sector workloads.

7.2.4 Data Center Architectures

Google, Sun, APC, and HP have all considered the possibility of a "data center in a container" [85, 7, 68, 71]. The approaches in this thesis are orthogonal to this design philosophy. Internet companies such as Google, Yahoo!, and Microsoft are designing Greenfield data centers [11, 12, 29]. The design of these data centers primarily focuses on novel solutions at a higher level, such as the design of hot/cold aisles, the use of ambient air for cooling, or the layout of power distribution units. The present work focuses on power and cooling efficiency at the server level, and would complement these Greenfield designs.

Chapter 8

Future Work

The work on memory disaggregation, compute disaggregation, and new server architectures opens up a number of rich new research areas. This chapter covers some of the possible avenues for future research on these topics, centered on improving efficiency and performance as well as increasing functionality.

8.1 Extending Memory Disaggregation

8.1.1 Policy Exploration and Other Workloads

One topic that has great importance to the performance of memory disaggregation is the policy for handling remote data. Comparing the baseline PS design versus the FGRA design, one of the biggest reasons for performance difference is that the PS design brings remote pages to the local memory space. For simplicity's sake, this work only explores using fairly standard techniques (Least Recently Used and Random) in the local-to-remote page eviction policy. Given the large benefit of correctly selecting the pages to be kept local, the exploration of more intelligent policies is a worthwhile endeavor. Policies that either keep track of page histories or identify hot versus cold pages may have performance benefits. This can be further augmented by related advances that can help

hide the latency of the memory blade, such as NUMA data movement, the critical block first optimization, and prefetching.

Beyond replacement policies, another area for future work is testing the performance of disaggregated memory on a greater variety of large-memory workloads. Studies are needed to examine the applicability of disaggregated memory for databases, especially including in-memory databases. Additionally, larger-memory variants of websearch and mapreduce would help demonstrate the memory blade's ability to maintain adequate performance workloads as they scale up. Further prototype development would be beneficial to enabling this large memory testing as detailed simulation incurs significant slowdowns.

8.1.2 Software Exposure of the Memory Blade

The design of disaggregated memory deliberately keeps the memory blade transparent to the underlying operating system to allow unmodified applications and OS's to utilize the expanded capacity. However, if the memory blade were exposed to the software through an application programming interface (API) such as POSIX, programmers would be able to take advantage of the multi-level memory hierarchy. The API would define the exact semantics for using the remote memory blade through function calls to the operating system. By using specific remote read and remote write calls, the programmer could explicitly store objects that are expected to be accessed less frequently on the remote memory blade, saving space in the local memory for more frequently accessed objects. For example, if a web search index can be divided into common and less common terms, the less common terms can be stored on the remote memory blade. Similarly, if the programmer has different performance requirements for different

sections of the application – a latency sensitive section versus a latency insensitive section – the objects can be split appropriately.

One further optimization may be to include multiple variations of access calls to the remote memory. Although only a single remote read call is likely to be needed, the remote write call could have a remote write temporary and remote write permanent to indicate the likelihood of the object being updated. This information can help if the memory blade has heterogeneous types of memory where one memory type may be more appropriate for long term storage. One example is Flash memory, which has significant write latencies but relatively fast read times, requires less power than normal DRAM, and is non-volatile.

To aid the memory blade in being exposed to the operating system, it would be useful to use the paravirtualization mode discussed in Section 5.3.1. This mode allows the operating system to make special hypercalls into the hypervisor to assist in virtualization. The standard hypercalls can be supplemented with new calls that directly send remote read and write requests to the hypervisor, allowing a fast-path access to the memory blade.

8.1.3 Memory Area Network

An additional extension of disaggregated memory is to utilize it for communication in a manner similar to shared memory. As mentioned in Section 3.2, support for distributed shared memory is not included on the memory blade. Doing so would require more complex interaction between the memory blade and the coherency protocol of the compute blade, potentially precluding the PS design. However, the design can be feasible if there are only specific portions of the memory blade that can be used for

communication of signals or objects by the transfer of ownership of regions of memory. In this case, extra programming interfaces to give and receive regions of memory would be required, but these interfaces only need hardware support at the memory blade itself. This communication through remote memory would create a “memory area network” among the compute blades connected to the memory blade. This network has the potential to speed up the latency and bandwidth of transfers compared to traditional packet networking, especially for applications that tend to frequently shuffle large amounts of data between servers, such as sort or MapReduce.

8.1.4 Memory Cloud

One further extension to the disaggregated memory design discussed in this thesis would be to expand the scope of the memory blades to span multiple enclosures. The current design has each memory blade service the compute blades within the same enclosure. Taking disaggregation a step further, one possibility would be to have an even larger memory resource that is shared by multiple compute blades across enclosures or even racks. This expanded scope would allow the larger memory pool to take even greater advantage of heterogeneous workload requirements across servers, and similarly allow the servers to utilize a larger memory pool than with a single memory blade. This design has additional implications on reliability; any outage of individual memory blades could be potentially remapped to the spare capacity of other blades in the memory cloud. Optical interconnects can make it possible to achieve the high connectivity needed with this large disaggregated memory at an adequate bandwidth and latency.

8.2 Synergy with Emerging Memory Technologies

Disaggregated memory adds a new layer to the conventional virtual memory hierarchy. This layer introduces several possibilities to integrate new technologies into the ensemble memory system that might prove latency- or cost-prohibitive in conventional blade architectures. The memory blade DRAM could be complemented or replaced with higher-density, lower-power, and/or non-volatile memory technologies, such as NAND Flash or phase change memory. Unlike conventional memory systems, where it is difficult to integrate these technologies because of large or asymmetric access latencies and lifetime/wearout challenges, disaggregated memory is more tolerant of increased access latency, and the memory blade controller can be extended to implement wear-leveling and other lifetime management strategies [50]. The previously discussed API extensions can also take advantage of the new memory types which have different access properties. Furthermore, disaggregated memory offers the potential for transparent integration. Because of the memory interface abstraction provided by the design, Flash or phase change memory can be utilized on the memory blade without requiring any further changes on the compute blade. This transparency can help facilitate the adoption of newer memory technologies into data centers.

8.3 Content-Based Page Sharing

Prior studies of consolidated VMs have shown substantial opportunities to reduce memory requirements via copy-on-write content-based page sharing across VMs [95]. Disaggregated memory offers an even larger scope for sharing content across multiple compute blades. By implementing content based sharing at the memory blade level, it is

possible to obtain significant capacity saving. Content-based page sharing can be implemented efficiently at the memory blade level by having an ASIC accelerator on-board that calculates hashes of page content as they are transferred to the memory blade. The accelerator would allow the memory blade to quickly check if the contents of the page being brought might match another page.

The two-level architecture of disaggregated memory offers unique opportunities for taking advantage of the different types of capacity. Optimizations can be geared towards utilizing the slower, but larger capacity of remote memory. One such optimization is in the virtualized desktop space. In this environment, users log in to their desktops which are running remotely on a server. This virtualization provides consolidation of users' desktops and easier management. Many of these desktops may be quite similar, offering potential for content-based sharing. Depending on the user workload, many desktops may be idle, yet their memory footprint will consume memory resources that could otherwise be used by more desktops.

One possible future optimization would be to leverage the memory blade and actively push the content of the idle desktops to remote memory; when combined with content-based page sharing, the movement of idle VMs to the memory blade would free up local memory to allow for even more consolidation of virtualized desktops onto a single server. A minimal amount of state of each desktop that is needed to respond to user input could be kept in local memory to minimize response time to wake up the idle desktop, and hide the latency of bringing its pages back to local memory.

8.4 Fast Virtual Machine Migration

Using the concept of the previously discussed memory area network, one future use for the memory blade could be for fast virtual machine migration. Typically when a virtual machine is migrated from one host to another, the entire memory contents of the VM must be transferred over the network. Based on the size of the VM and network speeds, this transfer may take tens of seconds, disrupting the running of the VM and taking up substantial network bandwidth [21]. There are “live” migration techniques that minimize the run-time disruption by iteratively copying the memory contents, trading off network bandwidth and total migration time to minimize down-time [21]. Memory disaggregation can provide an alternative to these approaches and provide fast VM migration by exploiting the shared nature of the memory blade. This migration could be done by having the ownership of any remote pages that are associated with a migrating VM be transferred between compute servers by simply changing a few mappings on the memory blade. This mapping change would save a significant amount of bandwidth and total migration time by not requiring any remote data to be copied. Such a mechanism could then either allow the rest of the VM’s local data to be transferred through normal means, or be moved to remote memory and then transferred through ownership changing. By being able to very quickly migrate VMs, this scheme may help enable new power and temperature-driven VM consolidation policies, allow aggressive server idling/shutdown, or quickly alleviate resource conflicts due to aggressive VM consolidation.

8.5 Memory Appliance Evolution

One natural direction for the memory blade is for it to become a full-fledged memory appliance that is capable of significant processing. Some possibilities include accelerators to provide compression, encryption, search matching, regular expression matching, sorting, prefetching, or hashing. By including an accelerator on the memory blade, commands could be explicitly sent to the blade and executed by it to assist the compute blades in processing data. The inclusion of an accelerator may have significant performance advantages, especially on larger data sets that require the capacity of the memory blade.

This idea can be taken one step further by having the memory blade include even more functionality. For example, one possibility is to use the memory blade as a staging area and DMA engine to transform data and transfer it to a remote location. In this scheme, compute blades could send the memory blade a command and data to encode; the memory blade would then transfer the encoded data to a secure storage service, such as Amazon's S3. Having distinct memory appliances with different functionalities can serve as potential building blocks in conjunction with the proposed memory cloud. Different memory appliances could be placed within a memory cloud to create unique and optimized ensembles of servers targeted towards specific mixtures of workloads.

8.6 Improved Warehouse-Computing Workloads

The benchmark suite described in Section 6.2 is meant to serve as a representative data point for warehouse-computing environments. However, it is only a single point on a continuum of behaviors, both within individual workloads and across data centers. Key

next steps for future work are to expand the benchmark suite to address these caveats and to validate the scaled-down benchmarks against real-world warehouse environments. Another important improvement is more complex cluster-level modeling that takes into account the interactions at a cluster-level as a workload scales or the number of servers scales.

8.7 Extending Compute Disaggregation

The proposed compute disaggregation separates large, monolithic server-class processors into smaller compute units using embedded-class processors. However, there are situations where more powerful processors are beneficial, as evidenced by the *webmail* results in Chapter 6. One possibility to extend compute disaggregation is an architecture where the smaller, disaggregated compute units can be coupled together to provide high, single system compute power when needed. Such an architecture would allow for flexible assignment of resources on an on-demand basis, allowing the compute resources to be properly provisioned for any workload. This disaggregated architecture would require high speed, coherent interconnects that can potentially travel large distances; optical interconnects may be able to provide a suitable interconnect. Additionally, cluster-wide control software would be required to enable dynamic provisioning of the resources. Previously discussed works such as Versatile SMP [80] may serve as the groundwork for the needed control software.

Chapter 9

Conclusions

The constraints on per-socket memory capacity and the growing contribution of memory to total datacenter costs and power consumption motivate a redesign of the memory subsystem. This thesis discusses a new architectural approach—memory disaggregation—which uses dedicated memory blades to provide OS-transparent memory extension and ensemble sharing for commodity-based blade-server designs. An extensible design is proposed for the memory blade which includes address remapping facilities to support protected dynamic memory provisioning across multiple clients, and power and density optimizations to address the compute-to-memory capacity imbalance. Two different disaggregated memory architectures are discussed that incorporate this blade: a page-based design that allows memory blades to be used on current commodity blade server architectures with small changes to the virtualization layer, and an alternative that requires small amounts of extra hardware support in current compute blades but supports fine-grained remote accesses and requires no changes to the software layer. This novel, commodity-based design simultaneously addresses compute-to-memory capacity extension and cross-node memory capacity sharing in a cost and power-effective manner. This thesis also considers dynamic memory sharing across the I/O

communication network in a blade enclosure and quantitatively evaluates design tradeoffs in this environment.

Simulations based on detailed traces from 12 enterprise benchmarks and three real-world enterprise datacenter deployments show that memory disaggregation has significant potential. The ability to extend and share memory can achieve orders of magnitude performance improvements in cases where applications run out of memory capacity, and similar orders of magnitude improvement in performance-per-dollar in cases where systems are overprovisioned for peak memory usage. This thesis also demonstrates how this approach can be used to achieve higher levels of server consolidation than currently possible.

A hypervisor-based prototype of disaggregated memory is developed to complement simulation and provide greater insight into the system-level implications of the new memory architectures. The primary functionality of the page-swapping design is implemented in Xen, a widely used and open-source hypervisor. The hypervisor is further modified to emulate having remote memory which has a different performance than local memory. Using this prototype, some of the results of the trace-based simulations are verified at a high level. Furthermore, the prototype highlights several system-level issues regarding hardware virtualization negatively impacting performance, software overheads associated with identifying remote pages, and super page mappings causing irregular performance.

At a broader level, disaggregated memory serves as an important building block for an architecture that significantly improves the efficiency of warehouse computing environments. Due to their massive scale, these data centers have unique processing,

power, and cost requirements. This thesis explores new server architectures based on the principles of disaggregation that holistically address these needs. Specifically these architectures utilize memory disaggregation; compute disaggregation through the use of compact, low-power and low-cost embedded class components; novel packaging and cooling optimizations; and flash-based disk caching along with low-power disks. Evaluated over a novel benchmark suite developed to model warehouse computing environments, these architectures have substantial improvements in performance-per-dollar efficiency compared to traditional servers. These architectures have had broad impact, and have led to a large body of work of efficiency-driven architectures for the internet sector.

Overall, as future server environments gravitate towards more memory-constrained and cost-conscious solutions, the memory disaggregation approach proposed in this thesis is likely to be a key part of future system designs.

References

[1] URL http://apotheca.hpl.hp.com/pub/datasets/animation-bear/.

[2] A. Agarwal, R. Bianchini, D. Chaiken, D. Kranz, J. Kubiatowicz, B. hong Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *In Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 2–13, 1995.

[3] AMD. AMD Geode LX Processor Family. URL http://www.amd.com/geodelx900.

[4] AMD. AMD Unveils Forward-Looking Technology Innovation To Extend Memory Footprint for Server Computing, July 2007. URL http://www.amd.com/us-en/Corporate/-VirtualPressRoom/0%2c%2c51_104_543~118446%2c00.html.

[5] C. Amza, A. L. Cox, H. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29: 18–28, 1996.

[6] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: a fast array of wimpy nodes. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 1–14, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-752-3. doi: http://doi.acm.org/10.1145/1629575.1629577.

[7] APC. InfraStruXure Express On-demand Mobile Data Center. URL http://-www.apc.com.

[8] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. COTSon: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43 (1): 52–61, 2009. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1496909.1496921.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006. URL http://-www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of*

the nineteenth ACM symposium on Operating systems principles, pages 164–177, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: http://doi.acm.org/10.1145/945445.945462.

[11] L. Barroso. Warehouse-scale Computers. Invited talk at the USENIX Annual Technical Conference, Santa Clara, CA, June 2007.

[12] L. Barroso, J. Dean, and U. Holzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23 (2), Mar./Apr. 2003.

[13] R. Bedichek. SimNow: Fast Platform Simulation Purely in Software. In *HotChips 16*, 2004.

[14] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne. Accelerating two-dimensional page walks for virtualized systems. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 26–35, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-958-6. doi: http://doi.acm.org/10.1145/1346281.1346286.

[15] V. P. E. T. Blog. Ten Reasons Why Oracle Runs Best on VMware. November 2007. URL http://blogs.vmware.com/performance/2007/11/ten-reasons-why.html.

[16] W. J. Bolosky, M. L. Scott, R. P. Fitzgerald, R. J. Fowler, and A. L. Cox. NUMA policies and their relation to memory architecture. In *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, pages 212–221, New York, NY, USA, 1991. ACM. ISBN 0-89791-380-9. doi: http://doi.acm.org/10.1145/106972.106994.

[17] R. Bryant. Data Intensive Super Computing. In *FCRC*, 2007. Keynote.

[18] R. Bryant and J. Hawkes. Linux Scalability for Large NUMA Systems. In *Linux Symposium*, 2003.

[19] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 217–228, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-406-5. doi: http://doi.acm.org/10.1145/1508244.1508270.

[20] Cisco. Cisco Unified Computing System Extended Memory Technology Overview, August 2009. URL http://www.cisco.com/en/US/prod/collateral/ps10265/ps10280/-ps10300/white_paper_c11-525300.pdf.

[21] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd*

conference on Symposium on Networked Systems Design & Implementation, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[22] M. D. Dahlin, O. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *In Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 267–280, 1994.

[23] J. De Gelas. AMD's 2010/2011 Roadmap from the IT Professional's Perspective, November 2009. URL http://it.anandtech.com/IT/showdoc.aspx?i=3681.

[24] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI-6*, 2004.

[25] F. Douglis. The Compression Cache: Using On-line Compression to Extend Physical Memory. In *In Proceedings of 1993 Winter USENIX Conference*, pages 519–529, 1993.

[26] DRAMeXchange. DRAM Price Quotes. URL http://www.dramexchange.com.

[27] M. Ekman and P. Stenstrom. A Cost-Effective Main Memory Organization for Future Servers. In *PDPS*, 2005.

[28] A. Falcon, P. Faraboschi, and D. Ortega. Combining Simulation and Virtualization through Dynamic Sampling. In *ISPASS*, 2007.

[29] X. Fan, W. Weber, and L. Barroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA-34*, 2007.

[30] M. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *In Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 201–212, 1995.

[31] E. W. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical report, Department of Computer Science and Engineering, University of Washington, March 1991.

[32] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004 (124): 5, 2004. ISSN 1075-3583.

[33] M. D. Flouris and E. P. Markatos. The Network RamDisk: Using Remote Memory on Heterogeneous NOWs. *Cluster Computing*, 2: 281–293, 1999.

[34] A. S. Foundation. Hadoop. URL http://lucene.apache.org/hadoop/index.html.

[35] I. Fried. Microsoft's data centers growing by the truckload, August 2008. URL http://news.cnet.com/8301-13860_3-10020902-56.html.

[36] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob. Fully-Buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling. In *13th International Symposium on High Performance Computer Architecture (HPCA 2007)*, pages 109–120, February 2007.

[37] K. Gharachorloo. The Plight of Software Distributed Shared Memory. In *Workshop on Software Distributed Shared Memory*, 1999.

[38] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *Internet Measurement Conference*, 2007.

[39] R. B. Gillett. Memory Channel Network for PCI. *IEEE Micro*, 16: 12–18, 1996.

[40] J. Hamilton. Cooperative Expendable Micro-Slice Servers (CEMS): Low Cost, Low Power Servers for Internet-Scale Services. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)*, January 2009.

[41] M. Hines, M. Lewandowski, and K. Gopalan. Anemone: Adaptive Network Memory Engine. Technical report, Florida State University, 2005.

[42] HP. Memory technology evolution: an overview of system memory technologies, April 2009. URL http://h20000.www2.hp.com/bc/docs/support/SupportManual/-c00256987/c00256987.pdf.

[43] L. Iftode, K. Li, and K. Petersen. Memory Servers for Multicomputers. IEEE Spring COMPCON ’93, 1993.

[44] S. M. Inc. Xeon 3000 1U Server. URL http://www.siliconmechanics.com/i7573/-xeon-3000-server.php.

[45] Intel. Intel Microprocessor Family Quick Reference Guide. . URL http://-www.intel.com/pressroom/kits/quickreffam.htm.

[46] Intel. Intel Server/Workstation Chipsets Comparison Chart. . URL http://-developer.intel.com/products/chipsets/index.htm.

[47] Intel. Intel Announces Next-Generation Atom Platform, December 2009. URL http://www.intel.com/pressroom/archive/releases/2009/20091221comp_sm.htm.

[48] C. Jenkins. In-memory databases. *ITNOW*, 43: 32, 2001.

[49] R. Katz. Research Directions in Internet-Scale Computing. Keynote presentation, 3rd International Week on Management of Networks and Services, 2007.

[50] T. Kgil and T. Mudge. FlashCache: a NAND Flash Memory File Cache for Low Power Web Servers. In *CASES’06*, 2006.

[51] T. Kgil, A. Saidi, N. Binkert, R. Dreslinski, S. Reinhardt, K. Flautner, and T. Mudge. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy

Efficient Chip Multiprocessor. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural*, pages 117–128, 2006.

[52] R. Khare, D. Cutting, K. Sitaker, and A. Rifkin. Nutch: A Flexible and Scalable Open-Source Web Search Engine. Technical Report CN-TR-04-04, CommerceNet Labs, Nov. 2004.

[53] S. Koussih, A. Acharya, and S. Setia. Dodo: A User-level System for Exploiting Idle Memory in Workstation Clusters. In *In Proc. of the Eighth IEEE Intl. Symp. on High Performance Distributed Computing (HPDC-8*, 1998.

[54] J. Larus. Spending Moore's Dividend. Technical Report MSR-TR-2008-69, Microsoft Research, May 2008.

[55] J. Laudon and D. Lenoski. The SGI Origin: a ccNUMA highly scalable server. In *24th International Symposium on Computer Architecture*, New York, NY, USA, 1997. ACM.

[56] A. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power Aware Page Allocation. In *ASPLOX-IX*, 2000.

[57] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-526-0. doi: http://doi.acm.org/10.1145/1555754.1555758.

[58] K. Leigh, P. Ranganathan, and J. Subhlok. General-purpose blade infrastructure for configurable system architectures. In *Distributed and Parallel Databases*, June 2007.

[59] D. Lenoski, J. Laudon, K. Gharachorloo, W. dietrich Weber, A. Gupta, J. Hennessy, M. Horowitz, M. S. Lam, and D. T. E. of use. The Stanford DASH multiprocessor. *IEEE Computer*, 25: 63–79, 1992.

[60] K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, 1989.

[61] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 315–326, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3174-8. doi: http://dx.doi.org/10.1109/ISCA.2008.37.

[62] J. Markoff and S. Hansell. Hiding in Plain Sight, Google Seeks More Power, June 2006. URL http://www.nytimes.com/2006/06/14/technology/14search.html.

[63] Micron. DDR2 Memory Power Calculator. URL http://download.micron.com/-downloads/misc/ddr2_power_calc_web.xls.

[64] Micron. Micron DRAM Memory Products, 2010. URL http://www.micron.com/-products/dram/.

[65] C. Moore. A Framework for Innovation. Keynote, FCRC, 2007.

[66] T. O'Reilly. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Sept. 2005. URL http://oreillynet.com/pub/a/oreilly/tim/news/-2005/09/30/what-is-web-20.html.

[67] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. DMA-Aware Memory Energy Management. In *In Proceedings of HPCA*, 2006.

[68] G. Papadopoulos. Redshift: The Explosion of Massive Scale Systems. Analyst Summit, 2007.

[69] C. Patel. Cooling arrangement for high performance electronic components, July 2002.

[70] C. Patel and P. Ranganathan. Enterprise Power and Cooling: A Chip-to-DataCenter Perspective. In *HotChips 19*, Aug. 2007. Tutorial.

[71] C. Patel and A. Shah. Cost Model for Planning, Development and Operation of a Data Center. Technical Report HPL-2005-107R1, Hewlett Packard Technical Report, 2005.

[72] PCIMG. Advanced TCA Specification. URL http://www.picmg.org/pdf/-PICMG_3_0_Shortform.pdf.

[73] PLX. PLX PCIe Switch Power Consumption. URL http:// www.plxtech.com/pdf/-technical/expresslane/Power_Consumption_Explained.pdf.

[74] Qualcomm. The Snapdragon Platform, 2009. URL http://www.qctconnect.com/-products/snapdragon.html.

[75] P. Ranganathan and N. Jouppi. Enterprise IT Trends and Implications for Architecture Research. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 253–256, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2275-0. doi: http://dx.doi.org/10.1109/HPCA.2005.14.

[76] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level Power Management for Dense Blade Servers. In *ISCA-33*, 2006.

[77] V. J. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web Search using Small Cores: Quantifying the Price of Efficiency. Technical Report MSR-TR-2009-105, Microsoft, August 2009. URL http://research.microsoft.com/apps/pubs/default.aspx?id=101821.

[78] J. Rolia, A. Andrzejak, and M. Arlitt. Automating Enterprise Application Placement in Resource Utilities. 2003.

[79] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.

[80] ScaleMP. The Versatile SMP (vSMP) Architecture and Solutions Based on vSMP Foundation. Technical report. URL http://www.scalemp.com/prod/technology/how-does-it-work/.

[81] D. J. Scales, K. Gharachorloo, and C. A. Thekkath. Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory. In *7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 174–185, 1996.

[82] I. Schoinas, S. K. Reinhardt, J. R. Larus, and D. A. Wood. Fine-grain Access Control for Distributed Shared Memory. In *In Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI*, pages 297–306, 1994.

[83] Seagate. Barracuda 7200.10 Data Sheet. URL http://www.seagate.com/docs/pdf/-datasheet/disc/ds_barracuda_7200_10.pdf.

[84] A. W. Services. Amazon Elastic Compute Cloud (Amazon EC2), 2009. URL http://aws.amazon.com/ec2/.

[85] S. Shankland. Google uncloaks once-secret server, April 2009. URL http://-news.cnet.com/8301-1001_3-10209580-92.html.

[86] SIA. International Technology Roadmap for Semiconductors 2007 Edition, 2007.

[87] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson. Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0. In *Cloud Computing and Its Applications*, October 2008.

[88] A. Sucharitakul. Benchmarking in the Web 2.0 Era. In *IISWC 2007*, 2007. Panel Session.

[89] T. M. Systems. TMS RamSan-440 Details. URL http://www.superssd.com/-products/ramsan-440/.

[90] A. S. Szalay, G. Bell, H. H. Huang, A. Terzis, and A. White. Low-Power Amdahl-Balanced Blades for Data Intensive Computing. In *2nd Workshop on Power Aware Computing and Systems (HotPower '09)*, October 2009.

[91] M. Technet. Tools for Performance Stressing Exchange 2003 Servers. 2003. URL http://technet.microsoft.com/en-us/library/aa996207.aspx.

[92] C. Thacker. Rethinking Data Centers. Invited talk at Stanford University Networking Seminar, Oct. 2007.

[93] P. Thibodeau. Google lands in Lenoir, N.C., as competition for data centers grows. *Computerworld*, Mar. 2007. URL http://www.computerworld.com/action/-article.do?command=viewArticleBasic&articleId=9015038.

[94] Virident. Virident's GreenGateway technology and Spansion EcoRAM.

[95] C. Waldspurger. Memory Resource Management in VMware ESX Server. In *OSDI-4*, 2002.

[96] M. Wu and W. Zwaenepoel. eNVy: A Non-Volatile, Main Memory Storage System. In *ASPLOS*, 1994.

[97] Y. Xie and D. O'Hallaron. Locality in Search Engine Queries and Its Implications for Caching. In *Infocomm*, 2002.

[98] D. Ye, A. Pavuluri, C. A. Waldspurger, B. Tsang, B. Rychlik, and S. Woo. Prototyping a hybrid main memory using a virtual machine monitor. In *IEEE International Conference on Computer Design*, 2008.