EXPLOITING PHASE-CHANGE TECHNOLOGY IN SERVER MEMORY SYSTEMS

BY LUIZ EDUARDO DA SILVA RAMOS

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Ricardo Bianchini

and approved by

New Brunswick, New Jersey

January, 2012

ABSTRACT OF THE DISSERTATION

Exploiting Phase-Change Technology in Server Memory Systems

by Luiz Eduardo da Silva Ramos

Dissertation Director: Ricardo Bianchini

Main memory capacity is becoming a critical issue for modern server systems. Unfortunately, current trends suggest that meeting these capacity requirements using DRAM will not be ideal. DRAM consumes significant amounts of energy (idle, refresh, and precharge energies) and will soon reach its density limit.

Many researchers in industry and academia point to Phase-Change Memory (PCM) technology as a promising replacement for DRAM. PCM is byte-addressable as DRAM, but presents higher density and lower idle power consumption than DRAM. However, PCM is also slower than DRAM and has limited endurance. For these reasons, hybrid memory systems that combine a small amount of DRAM and a large amount of PCM have become attractive.

In this dissertation, we propose two hybrid memory systems for servers. The first system (called Rank-aware Page Placement or RaPP) is a hardware-driven page placement policy. The policy relies on the memory controller (MC) to monitor access patterns, migrate pages between DRAM and PCM, and translate the memory addresses coming from the cores. The second system (called Rank-aware Cooperative Cache or RaCC) is a software-driven policy for object placement in server clusters that implement cooperative memory caches. RaCC monitors object popularity and leverages that information in placing the objects. Our extensive results show that our hybrid memory systems provide robust and consistent memory performance without sacrificing energy.

Based on our experience and results, we conclude that PCM is a promising main memory technology for future servers, especially when combined with a small amount of DRAM. However, such hybrid designs will require careful data placement and migration for best performance and robustness.

Acknowledgements

I would like to thank my mentor and friend Ricardo Bianchini for his many lessons, for his character, competence, patience, and generosity. I extend those acknowledgments to his family Marcia and Daniel.

I heartily thank all my past and present PhD committee members Abhishek Bhattacharjee, Ulrich Kremer, Bhuvan Urgaonkar, Badri Nath, and Andrew Nealen. In special, I also thank my professors Daniel A. Jiménez, Richard Martin, Thu Nguyen, Dinesh Pai, Endre Szemerédi, Liviu Iftode, and all our teaching staff.

I thank my lab mates from Dark and Panic Lab, specially Rekha Bachwani, Ana Paula Centeno, Guilherme Cox, Qingyuan Deng, Íñigo Goiri, Md. Haque, Taliver Heath, Kien Le, Cheng Li, Kiran Nagaraja, Fábio Oliveira, Chris Peery, Tuan Phan, Longhao Shu, Andrew Tjang, and Wei Zheng. They enlightened every "Dark moment".

I would like to thank our collaborators David Meisner and Thomas F. Wenisch (University of Michigan), and Eugene Gorbatov (Intel Corporation); our reviewers and people who somehow contributed to our work. Among those, we thank Luiz Barroso (Google), Frank Bellosa (University of Karlsruhe), Yung-Hsiang Lu (Purdue University), Stijn Eyerman (Ghent University), Yoshio Turner (HP Labs), and Hongzhong Zheng (University of Illinois at Chicago).

I sincerely acknowledge the Computer Science administrative and technical staff, as well as the CIFSS staff. Special thanks to those who I bothered the most throughout the years: Friedrick Crispin, Carol DiFrancesco, Maryann Holtsclaw, Hans Makmur, Charles McGrew, Mohini Mukherjee, Douglas Motto, Urmi Otiv, Robert Toth, Regina Ribaudo, Aneta Unrat, Joanne Walsh, and Donald Watrous.

I thank all my Computer Science colleagues and friends, in particular John McCabe, John-Austen Francisco, Jerry Hom, Rodrigo Toso, Daniela Viana, Wei Chen, Brian Russell, Brian Thompson, Desirée Ottoni, Christopher Woithe, Pravin Shankar, Lu Han, Reswana Nawrin, Mangesh Gupte, Gayatree Ganu, Konstantinos Kleisouris, Minji Wu, Pavel Kuksa, Monica Vroman, Robert Moore, Arzoo Zehra, Nikita Lytkin, Bruno Dufour, Carlos Diuk, Miguel Mosteiro, Nikolaos Leonardos, Cristopher Lauderdale, Shakeel Butt, Mohan Dhawan, Antonio Giovannelli, Enrique M. De Cote, Denitsa Tilkidjieva (in Memoriam), and Sérgio de Biasi (in Memoriam).

I would like to thank the United States for being my home during the PhD, and all the people and friends I was fortunate to meet here. I am very grateful to Diogo, Letícia, Gabriel, and Tomás Andrade and family, who literally adopted me into their household in the last months of my PhD. I also thank the very supportive friends: Tamanna Bhanot, Tibérius and Mara Bonates, Luís Castro, and family, Débora Espósito, Naomi Fleming, Michael Freitas, Lisa Giacumo, Aritanan Gruber, Slavko Komarnytsky, André and Eliane Madeira and family, Kelsie McClendon, Roberto Moratore Jr., Roberto, Vane, and Luisa Nicosia, Miguel Mosteiro and family, Carlos, Janaína and Lucas Oliveira, Guilherme Ottoni, Pooja Parek, Roy Pitta and family, Aline Ramires, Fernando Guerrero-Reyes, and so many others.

I am forever grateful to my parents Fernando and Sônia, my brother Paulo, and my entire family for their love and kindness. I love you all. I thank my dear wife Caroline for her priceless companionship, love and encouragement in critical times (I thank Sofia, Sky, and Jackson for the same reasons). I am also grateful to Caroline's family Anibal Savio, Maria Amalia Valiente, and Daniel Savio.

Finally, I would like to thank our sponsors: NSF (grant #CCF-0916539), Intel, CAPES (Brazil) and Fulbright. In particular, I am very grateful for support and patience of Silvio dos Santos Salles, Maria Luiza Lombas, and Jorge Pereira (CAPES), Michelle Lagos, Rosita Veluyia and Cinthya Alvarez (Fulbright), Glayna Braga, and Giselle Melo (Fulbright Brazil).

Dedication

To God, my parents Fernando and Sônia, my brother Paulo, and my wife Caroline.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Memory capacity is becoming a scarce resource in modern server systems, as the number of CPU cores increases, application and thread concurrency escalate, virtual machines require simultaneous co-location of large working sets in main memory, and data objects grow in size and quantity [28, 82]. Future projections suggest that at current rates, server systems will meet a "memory capacity wall", where memory capacity will limit the performance of servers [50, 63].

Unfortunately, current trends suggest that meeting the future capacity requirements using DRAM will not be ideal. DRAM exhibits low access times, but consumes significant amounts of energy (idle, refresh, and precharge energies). As a result, the amount of energy consumed by the memory is approaching (and sometimes surpassing) that consumed by the processors in many servers [4, 48]. Moreover, DRAM's storage elements are reaching their scalability limit. According to the latest ITRS report, DRAM will not scale below the 21nm technology node [37]. The alternative of adding memory modules is not cost-effective [10, 72], increases power consumption significantly [27, 55], and may be impossible due to design constraints (e.g., DDR3 supports at most two memory modules per channel) [51]. Thus, since DRAM is the dominant main memory technology, limiting DRAM chip densities ultimately jeopardizes the scalability of server memories.

For these reasons, architects have started to consider emerging memory technologies as potential replacements for DRAM [20, 24, 31, 47, 49, 85]. One of the most mature candidate technologies is Phase-Change Memory (PCM) [47, 67, 89, 90]. PCM is byte-addressable, consumes little idle energy, does not require refreshing or precharging (its contents are persistent), and exhibits access times in the nanosecond range.

Furthermore, PCM cells have feature size comparable to DRAM cells, but can store more information in the same physical area. However, PCM's read/write times, read/write energies, and write endurance are worse than those of DRAM.

1.1 Problem Statement

To overcome the limitations of DRAM and PCM, researchers have proposed hybrid memory systems that combine small amounts of DRAM and large amounts of PCM [59, 67, 89, 90]. Those systems can exploit the high performance of DRAM and the high capacity and scalability of PCM. However, for best performance and robustness, hybrid memory systems require careful management, as we demonstrate. In addition, existing approaches fail to leverage higher-level information available the workloads and services executed on servers. We demonstrate that high-level information can also benefit performance and robustness, while requiring lower complexity than existing hybrid memory systems.

1.2 In This Dissertation

In this dissertation, we propose two novel hybrid memory systems that combine DRAM and PCM. Both our systems seek to improve memory performance and robustness, without increasing energy consumption significantly. Our first design manages individual servers that run memory-intensive workloads. Our second design leverages service-level information in server clusters to attain better performance and robustness for I/O-intensive workloads. In fact, it entails the first study of the implications of PCM and hybrid main memory systems for server clusters.

1.2.1 Rank-aware Page Placement

Our first design comprises a sophisticated memory controller (MC) that implements a page placement policy called "Rank-based Page Placement" (RaPP). The policy efficiently ranks pages according to popularity (access frequency) and write intensity, migrating top-ranked pages into DRAM. For robustness, our MC may disable RaPP when it is unlikely to produce benefits.

We evaluate our memory system design and RaPP with regard to energy, performance, and endurance, using detailed simulations. For comparison with our system, we simulate two state-of-the-art hybrid memory designs [67, 89].

1.2.2 Rank-aware Cooperative Cache

Our second design targets server clusters that implement cooperative memory caches. These caches are extensively used in modern Internet services, and are a great target for hybrid memory systems, since they require low latency and high memory capacity at the same time.

Our design, called "Rank-aware Cooperative Cache" (RaCC), is a software-driven object placement policy for hybrid memory systems in server clusters. RaCC monitors object popularity and leverages that information in placing the objects across servers and memory technologies. Specifically, RaCC concentrates popular cached objects in the collective DRAM of the server cluster, while taking advantage of PCM's large capacity to increase the hit ratio of the cooperative cache. We apply RaCC to two state-of-the-art cooperative caches [12, 26].

We evaluate these RaCC-based systems with regard to energy, performance, and endurance, using simulation of real and synthetic traces of Internet services.

1.3 Contributions

In summary, our contributions in this dissertation are:

- We propose RaPP, a robust page placement policy for hybrid main memory systems;
- We evaluate RaPP and compare it to two state-of-the-art approaches;
- We study the implications of PCM and hybrid main memory systems for server clusters for the first time;

- We propose RaCC, a policy for managing object location in cooperative caches made from hybrid main memories; and
- We evaluate Memcached and PRESS, with and without RaCC, for PCM-based main memories.

We conclude that PCM is a promising main memory technology for future servers, especially when combined with a small amount of DRAM. However, such hybrid designs will require careful data placement and migration for best performance and robustness.

1.4 Dissertation Structure

The remainder of this dissertation is organized as follows. Chapter 2 provides background on server memory systems and main memory technologies. Chapters 3 and 4 present the details and evaluation of our hybrid memory systems. Chapter 5 discusses the related work and Chapter 6, contains our conclusions and future work.

Chapter 2

Background

In this chapter, we describe how architects typically organize main memory in server systems. We also provide deeper background on DRAM, discuss candidate technologies to replace DRAM (including PCM), and describe our assumptions for PCM.

2.1 Server Main Memory System

A server's memory system typically comprises a few key elements, namely a memory controller (MC), a few memory channels, and a number of dual-inline memory module (DIMMs). Each DIMM includes memory devices (chips) that contain a memory cell array and peripheral circuitry.

The MC is responsible for handling memory access requests from the CPU, resulting from last-level cache (LLC) misses or write-backs. The MC operates the memory chips to fetch and store data, as well as refresh their memory arrays (in the case of DRAM). Operating the memory chips entails forwarding physical addresses to the peripheral circuitry, and issuing commands to drive the chips while respecting certain timing constraints. Due to its central role, an MC can be designed with varying complexity to optimize the memory system for different purposes. For example, to improve access throughput, the MC may choose to buffer particular requests (e.g., LLC write-backs) and/or reorder requests according to the system's current state [5, 39, 71].

At a high level, a memory chip can be logically viewed as an array of bit cells with some interface circuitry. Memory chips are read and written in groups called ranks. A rank can transfer a number of bits equal to the memory channel's width (e.g., 64 bits) in one memory cycle, so it takes several memory cycles to transfer an entire LLC line. At a low level, the memory chips are subdivided into banks that can be accessed concurrently, and further into bitcell arrays. On a read to a bank, part of the target address (provided by the MC) is used to activate a row of bitcells, and amplify and latch their contents into a row buffer. Then, a column (subset of a row) can be selected, causing its data to be transferred from the row buffer to the MC. The MC then delivers the data to the CPU. An access to a column of another row causes the activation of the new row. On a LLC write-back, the data flows in the opposite direction, is stored in the row buffer, and eventually written to the memory array.

2.1.1 Synchronous Dynamic RAM (SDRAM)

A DRAM cell is a transistor-capacitor pair (1T/1C) where each transistor allows reading and writing, and the capacitor stores a bit as an electrical charge (Figure 2.1). Since DRAM's capacitors discharge over time, they need to be refreshed at regular intervals to prevent data loss. In addition, reading a DRAM cell destroys its original content; the "precharge" operation restores the data from the row buffer into the array. Under a close-page management scheme, the MC precharges (closes) a row after every column access, unless there is another pending access for the same row. If there is a pending access, the row buffer remains open and the access can be serviced from there. Under open-page management, the row buffer remains open until another row needs to be loaded into it; only at that point is the precharge operation performed. Close-page management typically works better than open-page management for multi-core systems [77].

Double Data Rate 3 (DDR3) is the current standard for Synchronous DRAM interfaces. DDR3 channels are 64-bit wide (72-bit wide with ECC), so it takes four memory cycles (8 transfers) to access a 64-byte LLC line. In server systems, typically one, two, or four ranks of 8 (x8) or 16 (x4) chips (plus additional chips when using ECC) are laid out on each DIMM. When designed to operate at high frequencies, only one or two DDR3 DIMMs can lie on a single memory channel due to electrical limitations.

DRAM energy can be broken down into background, activation/precharge, read/write, and termination energies. Background energy is independent of activity and

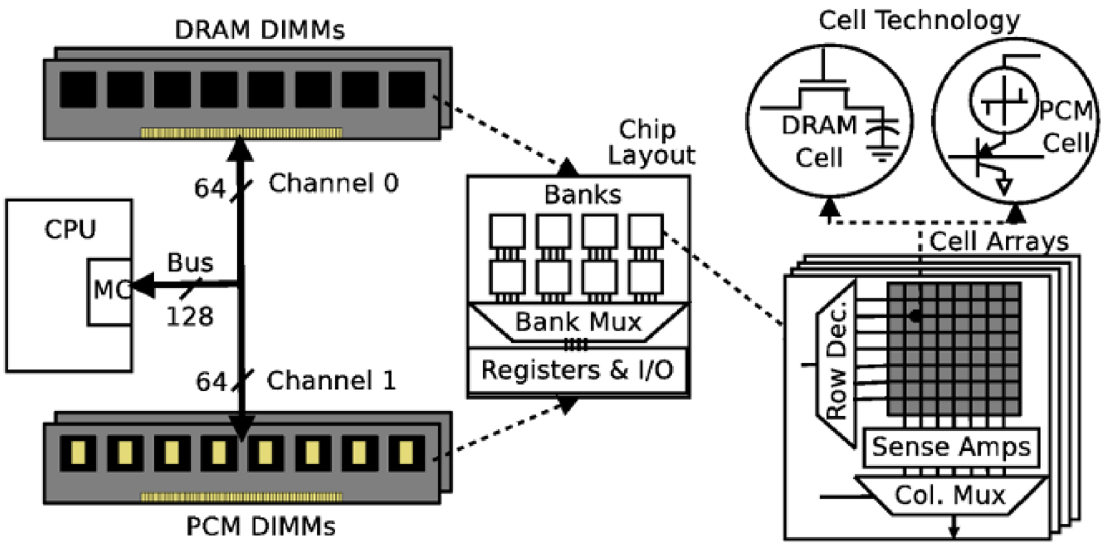


Figure 2.1: Memory system using DRAM and PCM DIMMs. DRAM and PCM arrays are slightly different (cells and sense amplifiers).

is due to the peripheral circuitry (e.g., row decoder, column muxes, sense amplifiers, and bus drivers), transistor leakage, and refresh operations. The activation/precharge energy is due to these two operations on the memory arrays. The read/write energy is due to column accesses to row buffers. The termination energy is due to terminating data and strobe signals at each chip, as well as signals of other ranks on the same channel. The three latter classes are often referred to as “dynamic DRAM energy”. Most of the energy consumed by DRAM is background and activation/precharge energy.

2.2 Candidate Replacements for DRAM

For over a decade, DRAM has been the dominant main memory technology. Its adoption was fueled by its high scalability, energy-efficiency, and low cost per bit compared to Static RAM, and high performance compared to Flash memory [36]. Unfortunately, the DRAM storage element will soon reach its scalability limit [37,38]. Physical space constrains the memory design mainly because cell capacitance becomes too low (and leakage power too high), requiring impractical refresh rates [37]. The latest ITRS report (2009 with an update in 2010) mentions stack capacitor cells as a workaround that enables the non-scaling DRAM capacitors to be used in smaller

DRAM cells. That solution entails non-planar structures with high aspect ratio and smaller size ($4F^2$ instead of $6F^2$). However, that solution has its own problems and should enable DRAM to scale further down to somewhere between 21 and 18nm feature sizes [38], beyond which this technique cannot scale further. Without the scalability advantage and with increasing contribution of main memory to capital costs and power consumption in servers, architects have been searching for alternative memory technologies to replace DRAM. Table 2.1 compares DRAM to other memory technologies.

Table 2.1: Comparing DRAM [36, 37, 55], FeRAM [20, 24, 43], STT-RAM [30, 58, 85], PCM [47, 67], and Flash [14, 56, 80].

Features	DRAM	FeRAM	STT-RAM	PCM	NOR Flash	NAND Flash
Cells size (F^2)	6 – 8	15–32	36	4 – 12	10	4
Erase block / latency	N.A.	N.A.	N.A.	N.A.	64–256KB / 900ms	8–64KB / 2ms
R / W page size	64b	64b	64b	64b	64b	512b–4KB
Read latency	10's ns	10's ns	10's ns	10's ns	100's ns	10's μs
Write latency	10's ns	10's ns	10's ns	100's ns	10's μs	100's μs
Non-volatility	no	yes	yes	yes	yes	yes
Endurance	10^{16}	10^{14}	10^{15}	10^8–10^9	10^5	10^5

Despite the advanced maturity level of NOR and NAND Flash, they are unsuitable for main memory. NOR Flash is byte-addressable for reads and writes, but does not support in-place overwrite of a modified bit without first erasing the block (typically 64–256KB) that contains that bit. The erase procedure must occur for the entire block simultaneously and takes milliseconds to seconds to complete [14]. NAND Flash is slower than NOR for reads and writes, but erases faster. NAND also has higher density than NOR and is page-oriented (512b–4KB reads, writes, and erases), which makes it more suitable for storage and less suitable for main memory. In addition to these features, both NOR and NAND Flash have very poor endurance (10^5 erases, or less for feature sizes below 25nm).

Due to the limitations of DRAM and Flash, Storage Class Memories (SCM) have emerged as possible future replacements, and have been gaining increasing attention. Three of the most mature SCMs are: Ferroelectric RAM (FeRAM), Spin-Torque Transfer RAM (STT-RAM), and Phase-Change Memory (PCM). These technologies are currently under development and first-generation devices have already been used in embedded systems [20,25,61,79]. However, they have not been developed commercially for main memory at this point.

In particular, FeRAM and STT-RAM are promising replacements for SRAM (e.g., in caches, CPUs and embedded systems) [30, 58, 78, 85]. However, they have shown somewhat limited scalability, high cost and integration issues [37, 38], which hinder their use in main memory. In contrast, PCM devices of higher density have already been produced [73] (512Mbit vs 4Mbit of FeRAM and STT-RAM), and the current roadmap depicts PCM scaling more aggressively than the other two [37]. In fact, one recent research work [1] has developed a prototype of PCM DDR2 DIMM 250MHz (using Micron's 128Mbit chips [57]) and a memory controller that implements the wear-leveling algorithm in [69]. The DIMMs are, however, used in a storage device attached to the PCIe interface. For all these reasons, multiple works, including this dissertation, have focused on the PCM SCM. In the next section, we describe the PCM cell and interface in more detail.

2.2.1 Phase-Change Memory

A PCM cell comprises an NMOS access transistor and a storage resistor (1T/1R) made of a chalcogenide alloy. With the application of heat, the alloy can be transitioned between physical states with particular resistances, used to represent binary values. When the alloy is heated to a very high temperature ($> 600^oC$) and quickly cooled down, it turns into an amorphous glass with high electrical resistance, representing 0. When the alloy is heated to a temperature between the crystallization (300^oC) and melting (600^oC) points and cools down slowly, it crystallizes to a state with lower resistance, representing 1. This programming process can be carried out by peripheral write drivers.

A cell's content can be read using current sense amplifiers to measure its electrical resistance, as opposed to DRAM's slower but smaller voltage sense amplifiers. PCM can thus interface with most CMOS peripheral circuitry used in DRAM [89]. Unlike in DRAM, PCM's reads are non-destructive and cells can retain data for several years.

On the downside, PCM cells exhibit worse access performance than DRAM.

Regarding density, PCM has similar cell size ($4F^2$ to $12F^2$) compared to DRAM ($6F^2$ to $8F^2$). However, PCM enables manufacturing of multi-level cells (MLC), which produce intermediate resistances (the alloy is partially crystalline and partially amorphous) and therefore can store multiple bits. Current MLC prototypes have two- or four-bit cells, capable of storing four and sixteen binary values, respectively [62]. Assuming the same cell size for both technologies, these MLCs hold twice and eight times more data than DRAM cells in the same area.

Our assumptions for PCM. In this dissertation, we assume that PCM is four times more storage-dense than DRAM, as in [67]. For easier adoption, we expect that the peripheral circuitry for PCM (e.g., row buffers, row and column decoders, DIMM interface) will be equivalent to that for DRAM, except for sense amplifiers. Thus, we assume this circuitry to have the same performance and power characteristics for both PCM and DRAM. Previous studies have made the same assumption [47,89]. Only the written cache lines in a row buffer are written back to the PCM cell array (DRAM needs the entire buffer to be written back to precharge its cells). Similar optimizations have been used before as well [47, 90]. To expose the entire overhead of PCM accesses to the cores, we study a CPU with in-order cores and a single outstanding miss per core.

PCM does not require cell refreshing or precharging, thereby lowering background energy relative to DRAM and eliminating precharge energy. However, PCM increases activation and termination energies, since its activations (actual accesses to memory cells) are slower than with DRAM. Our assumptions for peripheral circuitry imply that row buffer read/write energy is the same for DRAM and PCM.

Chapter 3

Rank-aware Page Placement

3.1 Overview

In this chapter, we present a new DRAM+PCM memory system design that is robust across a wide range of workloads. The design comprises a sophisticated MC that implements a page placement policy called RaPP. The policy efficiently ranks pages according to popularity (access frequency) and write intensity, migrating top-ranked pages to DRAM. While monitoring popularity, RaPP penalizes pages that are unlikely to produce benefits if migrated. To improve PCM's endurance, each migration involves two PCM memory frames and one DRAM frame. The MC monitors access patterns and, when necessary, migrates pages. The migrations are not immediately visible by the OS, as the MC uses its own address translation table. Periodically (or when the table fills up), the OS updates its mapping of virtual pages to physical frames based on the translation table and clears it.

We evaluate our memory system design and RaPP using a detailed simulator that computes the energy, performance, and endurance of workloads running on an 8-core CPU. For comparison with our system, we simulate two state-of-the-art hybrid memory designs [67, 89], as well as a baseline hybrid system without page management (called "unmanaged") and a PCM-only system.

Our results for 27 workloads show that our system consumes roughly the same amount of power on average as its competitors and the baselines, but with significantly better performance. In terms of energy-delay^2, for the workloads we study, our system is on average 36% better than the PCM-only baseline and 24% better than the unmanaged hybrid system. Compared to the state-of-the-art hybrid systems, our system exhibits at least 13% better energy-delay^2 on average, especially for workloads with large memory footprints. Our system also improves lifetime, as compared to the baselines, but not enough to enable system operation for 5 years assuming current PCM endurance. Nevertheless, PCM endurance is expected to increase by orders of magnitude in the next few years [17, 36]. Until then, our system can be easily combined with previously proposed endurance-improvement techniques [16, 40, 47, 90].

3.2 Hybrid Main Memory Design

As we mentioned in Section 1, given the speed of DRAM and the high density of PCM, there is a clear incentive for combining these two technologies into a single, hybrid memory system. However, PCM has undesirable characteristics (poor performance, dynamic energy, and endurance) that must be properly managed. Similarly, DRAM has a relatively high idle energy consumption compared to that of PCM.

A few previous works have realized the benefits of combining DRAM and PCM [67, 89] and the associated tradeoffs. Unfortunately, the previous approaches have serious limitations. Qureshi *et al.* [67] proposed to use DRAM as a buffer in front of PCM. Since the buffer is managed as an inclusive hardware cache, the DRAM space does not add to the overall memory capacity. More importantly, for workloads with poor locality, the cache actually lowers performance and increases energy consumption. Zhang *et al.* [89] combine the DRAM and PCM areas in a large flat memory and migrate pages between the areas. However, the migrations are performed by the OS and target the frequently written pages, leaving read-intensive pages in the slower PCM area.

Next, we describe our hardware-software page placement policy, called RaPP, which manages pages without the limitations of previous works. After that, we detail our implementation of the two prior hybrid systems. Throughout our descriptions, we differentiate between *virtual memory* pages (or simply pages) and *physical memory* frames (or simply frames).

3.2.1 Rank-based Page Placement

Given the characteristics of DRAM and PCM, RaPP seeks to (1) place performance-critical pages and frequently written pages in DRAM, (2) place non-critical pages and rarely written pages in PCM, and (3) spread writes to PCM across many physical frames.

The justification for these goals is that previous works have shown that typically only a relatively small subset of pages is performance-critical during the execution of a workload [33]. This observation suggests that (a) this subset may fit entirely in the DRAM part of the hybrid memory, and (b) the majority of lightly accessed pages should consume little energy, if stored in the PCM part. Moreover, previous work has found that the subset of critical pages may change over time, along with the criticality of individual pages [33]. This second observation suggests that the system must dynamically identify the critical pages and adjust their placements accordingly.

Since the OS is not on the path of most memory accesses, RaPP must be collaboratively executed by the OS and the MC. An interesting challenge is that neither the MC nor the OS has complete information about the performance criticality of the pages in a workload. For example, the latency of the cache misses associated with a page may be hidden behind out-of-order execution or multithreading by the processor cores. Interestingly, previous work [6] has shown that the frequency of cache misses is a very good proxy for a thread's performance criticality, regardless of the details of the microarchitecture (in-order vs out-of-order execution). Thus, pages that experience more misses also tend to be more performance critical.

RaPP relies on the MC to monitor the misses in the LLC to each physical memory frame. In addition, the MC monitors the LLC write-backs directed to each frame. Using this information, RaPP dynamically ranks frames based on frequency and recency of accesses, as detailed below. Frames that rank high are called "popular", and frames that rank low are called "unpopular". Whenever the most popular PCM frame reaches a threshold number of accesses (called the "migration threshold"), the MC considers migrating its content into DRAM transparently to the OS. If the DRAM area is full, the MC selects the page stored in an unpopular DRAM frame to migrate to PCM.

Ranking frames. RaPP simultaneously considers frame access frequency and recency in its dynamic ranking of pages (i.e., the pages stored in the frames), using a modified version of the Multi-Queue (MQ) [91] algorithm for second-level buffer cache replacements. As originally designed, MQ defines M LRU queues of block descriptors, numbered from 0 to $M-1$. Each descriptor includes the block number, a reference counter, and a logical expiration time. The descriptors in queue $M-1$ represent the blocks that are most frequently used. On the first access to a block, its descriptor is placed in the tail of queue 0. In addition, the block's expiration time $ExpirationTime$ is set to $CurrentTime + LifeTime$, where both times are measured in number of accesses and $LifeTime$ specifies the number of consecutive accesses that must directed to other blocks before we expire the block. Every time the block is accessed, its reference counter is incremented, its expiration time is reset to $CurrentTime + LifeTime$, and its descriptor is moved to the tail of its current queue. The descriptor of a frequently used block is promoted to a higher queue (saturating at queue $M-1$, of course) after a certain number of accesses to the block. Specifically, if the descriptor is currently in queue i, it will be upgraded to queue $i+1$ when its reference counter reaches 2^{i+1}. Conversely, MQ demotes blocks that have not been accessed recently. On each access, the descriptors at the heads of all M queues (representing the LRU block of each queue) are checked for expiration ($CurrentTime > ExpirationTime$). If a block descriptor expires, it is placed at the tail of the immediately inferior queue, and has its expiration time again set to $CurrentTime + LifeTime$. Figure 3.1 shows an example of promotion and demotion in MQ.

We use the modified MQ to rank memory frames (it was originally designed to rank disk blocks). We do so for two main reasons: (1) as page migrations are expensive operations, it is important to select the pages to migrate as intelligently and accurately as possible. MQ has been proven superior to other algorithms in selecting the blocks to replace [91]; (2) modern memory controllers are becoming increasingly complex and sophisticated (as discussed below), as a result of the increasing importance of the

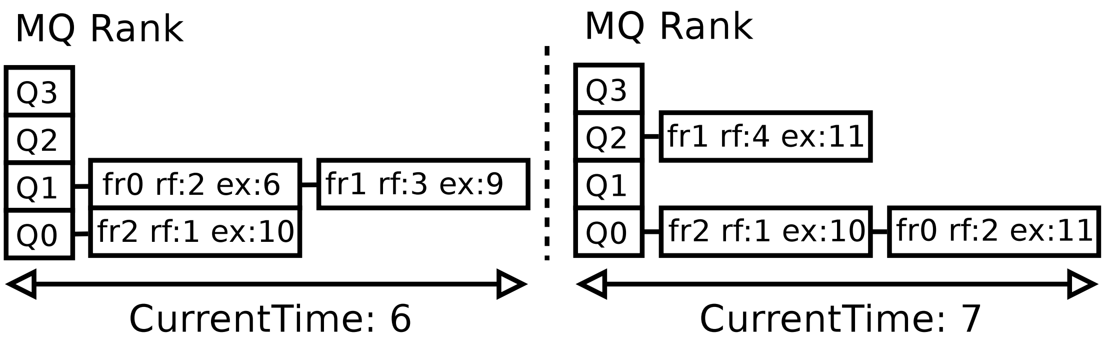


Figure 3.1: MQ example with M=4 and *LifeTime*=4. At *CurrentTime* 6, the rank contains three frames, represented by a tag (fr0-2), a reference counter (rf) and an *ExpirationTime* (ex). When fr1 receives a reference at *CurrentTime* 7, it is promoted into queue 2, whereas fr0 gets demoted to queue 0.

memory system (in terms of performance and energy) and relentless technology scaling. To avoid performance degradation, *the updates to the MQ queues are performed by the MC off the critical path of memory accesses,* using a separate queue of updates and a small on-chip SRAM cache. To find the MQ entry of a frame, the MC hashes the corresponding frame number.

We create 15 queues (numbered 0–14) plus a 16th victim list (described below). Pages stored in PCM frames that become popular (i.e., get to higher queues) are scheduled for migration to DRAM. However, we modified MQ in two important ways. First, instead of counting all accesses, we only count an access if it occurs more than a threshold time (measured in memory cycles) after the last access to the same frame. This latter threshold is called the "filter threshold". The MC stores the time of the last access in the descriptor for the frame. The reason for filtering rapid-fire accesses out is that there is no point in trying to migrate a page that is accessed in such a way; before we get to migrate the page, the needed data has already been loaded to the LLC (or evicted from it). In fact, it is possible that the page will not even be accessed again in memory. Using a 2-competitive approach, we set the filter threshold to be $MigrationCost/MigrationThreshold$, where $MigrationCost$ is the uncontended number of memory cycles needed to migrate a page. ($MigrationCost$ is roughly 1.6μs in our experiments.)

Second, we modified the demotion policy in the following ways: (a) we use time, not number of accesses, as the metric for demotion to reduce space requirements (in our experiments, we set $LifeTime$ to 100μs, which works well for our workloads); (b) we only demote from one queue at a time (in round-robin fashion) to reduce runtime overhead; and (c) a DRAM frame that is demoted twice without any intervening accesses leaves the MQ queues and becomes a candidate to receive a popular PCM page. The reason for the latter modification is that frames that undergo multiple demotions tend to have already been cached in the LLC and will not be accessed in a while. We store the MQ queues and the victim list in the lowest DRAM addresses.

Migrating pages. As mentioned above, RaPP schedules the page stored in a PCM frame for migration to DRAM after its reference counter reaches the migration threshold. In particular, the page stored at any PCM frame that reaches queue 5 (i.e., the reference counter for the frame has reached $2^5 = 32$) is scheduled for migration to DRAM. (Thus, the maximum number of frames that can be in queues 5–14 is the size of DRAM. For symmetry, the maximum size of queues 0–4 is also set to the size of DRAM.)

We find these values for M and the migration threshold to work well for our extensive set of workloads. The rationale is that in many workloads, a large number of pages would end up in queue $M - 1$ if M is small, compromising the accuracy of the hot page identification. On the other hand, if M is high, RaPP can correctly identify hot pages, but the MQ overhead increases. As for the migration threshold, we must select a value that enables early migrations but without migrating pages unnecessarily.

To select a destination DRAM frame for a page, the MC maintains an LRU list of victim DRAM frames. The victim frames are not in any of the LRU queues (the list is initialized with all DRAM frames). Whenever a frame on the victim list is accessed, it is removed from the list and added to queue 0. A frame demoted from queue 0 or demoted twice without intervening accesses is moved to the tail of the list. The destination DRAM frame is the first frame on the list. If the list is empty, no destination is selected and the migration is delayed until a frame is added to the list.

To effect a page migration to DRAM, the MC (1) migrates the page stored in the selected DRAM frame to one of the unranked PCM frames, (2) migrates the content of this latter frame to the most popular PCM frame, and finally (3) migrates the content of the most popular PCM frame to the selected DRAM frame. Figure 3.2 shows an example, where shaded frames are PCM frames and non-shaded frames are DRAM frames. In the example, the MC migrates the content of frame 13 to frame 2, the content of frame 2 to frame 19, and the content of frame 19 to frame 13.

To allow the three migrations to proceed concurrently, the MC uses three intermediate frame-size buffers located in the MC itself. The contents of the frames are first copied to the buffers, and only later copied to the destination frames. In addition, to avoid excessively delaying LLC misses due to row conflicts while migrating, the PCM DIMMs are equipped with an extra pair of row-buffers per rank, used exclusively for migrations. Operated by the MC, these buffers communicate with the internal prefetching circuitry of the PCM DIMM [21, 22], bypassing the original bank's row buffer. Since our migrations occur in sequence, two of these buffers are necessary only when the migration involves two banks of the same rank, and one buffer would suffice otherwise. This modification is not applied to DRAM DIMMs to avoid their redesign. (The energy and delay costs of these extra PCM DIMM buffers are taken into account in our simulations.)

RaPP uses a different destination unranked (and thus unpopular) PCM frame every time it needs to migrate a page out of DRAM. The reason is that migrations involve writes to the PCM cells. Using different unpopular pages guarantees that these writes are evenly spread across the PCM area for wear leveling. We start picking unranked frames from the bottom of the physical address space (which maps to the end of the PCM area), and move upward from there whenever a new PCM frame is needed.

The set of scheduled migrations is maintained in a list. We deschedule migrations whenever the corresponding PCM pages cross back down to queue 4 before the migrations start. The MC performs migrations from the list whenever there are no LLC misses or write-backs to perform. Any misses that arrive for a page undergoing a migration are directed to the original address or to one of the intermediate buffers. Write-backs are buffered until the migration is concluded.

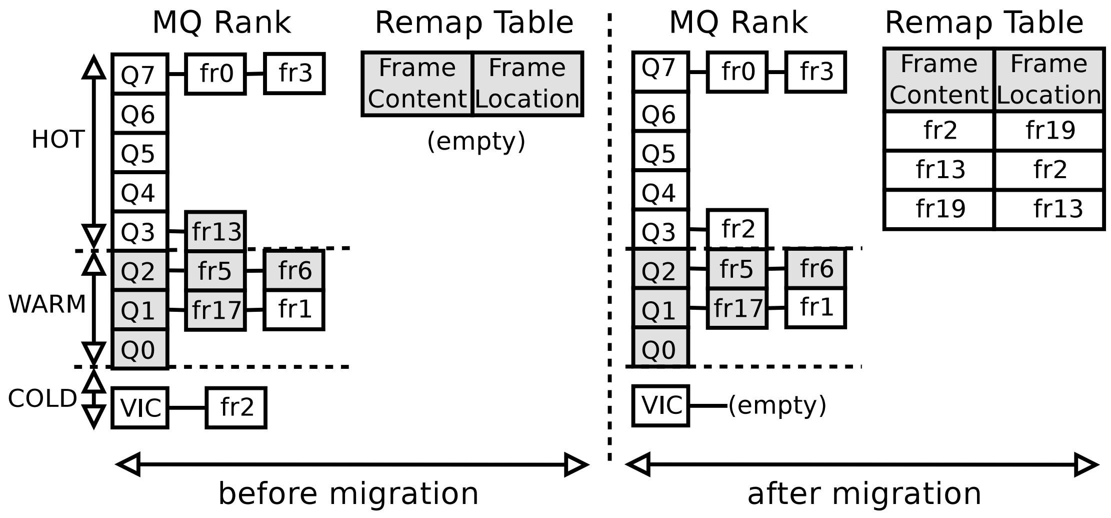


Figure 3.2: RaPP example. Before and after we migrate a PCM page (stored in frame 13) that crossed over to the DRAM side of the ranking to a frame from the victim list (frame 2). Dark shading represents PCM frames and no shading represents DRAM frames.

For best performance, our goal is to execute the migrations completely in the background and without OS involvement. Thus, the MC maintains the *RemapTable*, a hash table for translating frame addresses coming from the LLC to actual remapped frame addresses. Figure 3.2 shows an example *RemapTable*. The *RemapTable* is accessible by the OS as well. Periodically or when the *RemapTable* fills up (at which point the MC interrupts the CPU), the OS commits the new translations to its page table and invalidates the corresponding TLB entries. (When non-virtually addressed hardware caches are used, some lines may have to be invalidated as well.) We assume that the OS uses a hashed inverted page table, as in the UltraSparc and PowerPC architectures, which considerably simplifies the commit operation. Since a commit clears the *RemapTable*, the OS sets a flag in a memory-mapped register in the MC to make sure that the MC refrains from migrating pages during the commit process.

The *RemapTable* also includes two bits for communication between the MC and the OS. One bit is called *MigratingNow*, which when set means that the corresponding frame is currently scheduled for a migration. The other bit is called *ReplacingNow*, which when set means that the OS is replacing the page currently stored in that frame. The MC is responsible for *MigratingNow*, whereas the OS is responsible for

ReplacingNow. Before the OS tries to replace a page, it must check the *RemapTable* first. There are three possible scenarios here. Scenario 1: If there is no entry for the physical frame in which the page lies, the OS creates one, sets *ReplacingNow*, and programs the DMA engine to use the frame. The MC does not migrate any page to that same frame while *ReplacingNow* is set. When the replacement is done, the OS resets *ReplacingNow*. Scenario 2: If there is an entry for the corresponding frame and *MigratingNow* is set, the OS should select another page for replacement. Scenario 3: If the frame has already changed addresses (i.e., the entry for the frame exists and *MigratingNow* is not set), the OS can set *ReplacingNow* and proceed using the new frame address.

Finally, for robustness, RaPP uses a self-disabling mechanism that disables access monitoring, queue maintenance, and migrations whenever too many "bad migrations" occur. A bad migration occurs in one of two cases: (1) when a page originally in PCM is migrated to DRAM and then back to PCM without being referenced enough times while in DRAM; or (2) when a page originally in DRAM is evicted to PCM and then back to DRAM with too many accesses while in PCM. To implement this mechanism, we use a single counter of bad migrations (CBM) and a 2-bit saturating counter per MQ entry. Whenever a ranked page is touched, the saturating counter is incremented. Whenever a migration is completed, using the *RemapTable*, RaPP can identify where the page was since the last commit to the OS page table. If the migration falls into case (1) and the counter is not saturated, or it falls into case (2) and the counter is saturated, CBM is incremented. The saturating counter for a page is reset whenever the page migrates. At the end of each 1ms epoch, if the number of bad migrations reached 5% (the "disable threshold") or more of the maximum number of migrations possible within the epoch, RaPP is disabled.

Controller structure. Our MC adds a few components to a vanilla MC. Our MC (Figure 3.3) extends a programmable MC from [86] by adding RaPP's own modules (shaded in the figure). The MC receives read/write requests from the LLC controller via the CMD queue. The Arbiter iteratively dequeues requests, which the Controller

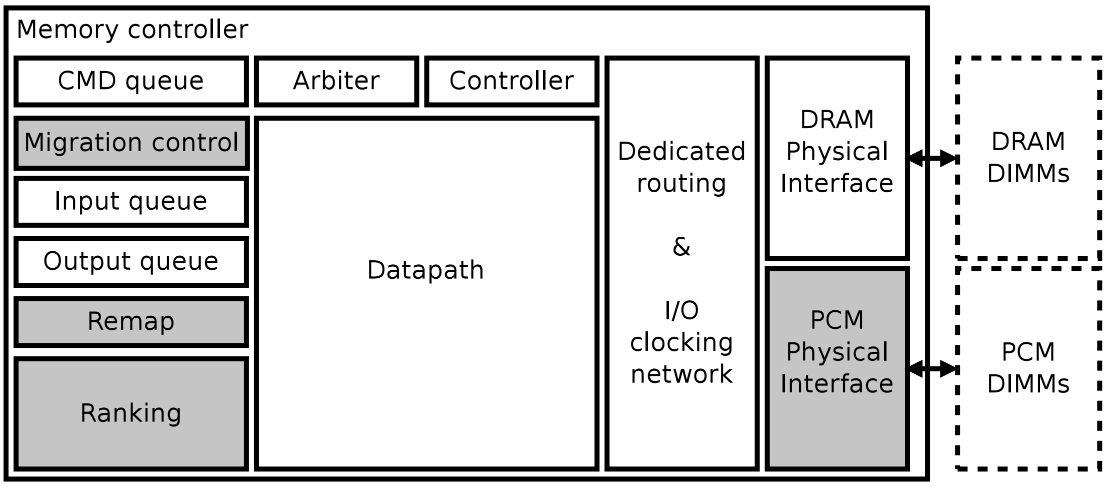


Figure 3.3: Memory controller with RaPP's new modules highlighted.

converts into commands to be sent to the memory devices. The Controller routes those commands by frame address to the appropriate Physical Interface (DRAM or PCM), which converts commands into timing relationships and signals for operating memory devices and coordinating data transfers. The Interfaces also control the power states of their respective memory ranks. The Datapath handles the data flow from the memory devices to the LLC controller and vice-versa. The module places data read from memory into the Output queue, where the LLC controller can read it. On write-back requests, the Datapath reads data (provided by the LLC controller) from the Input queue. For consistency, the CMD queue logic checks if the target address of a read or write-back collides with older write-backs in the Input queue. A colliding read is serviced from the queue without actually reaching the memory devices, thus finishing faster. A colliding write-back invalidates the older write-back command and data.

RaPP's Ranking module (RKMOD) contains the small on-chip cache of MQ and victim entries (entry cache) and the queue for updates to the MQ queues and the victim list (update queue). Misses in the entry cache produce requests to DRAM. RKMOD's logic snoops the CMD queue, creating one update per new request. To reduce the lag between an access and its corresponding MQ entry update, the update queue is implemented as a small circular buffer (32 entries), where an entering update precludes any currently queued update to the same entry.

The Migration Control module (MIGMOD) contains the queue of scheduled migrations (migration queue) and three page-sized buffers for the migrations (transfer buffers). MIGMOD processes migrations sequentially, each one occurring in two stages: (1) read and buffer frames; and (2) write frames to their new locations. Stage (2) does not start until stage (1) is completed. MIGMOD latches the base addresses of the frames undergoing a migration, as well as an offset address within each frame. The Controller module detects memory accesses that target one of the frames undergoing migrations by checking the base addresses. The Controller serves an access that targets a migrating frame by accessing the appropriate structure (main memory or a transfer buffer).

The Remap module (REMOD) contains the *RemapTable* and the logic to remap target addresses. At the end of a migration, MIGMOD submits the three latched frame numbers to REMOD, which creates new mappings in the *RemapTable*. REMOD snoops the CMD queue to check if it is necessary to remap its entries. Each *RemapTable* lookup and each remapping take 1 memory cycle. However, these operations only delay a request if it finds the CMQ queue empty.

Note that many previous works have proposed MCs with similar levels of sophistication and complexity, e.g. [19,23,39,40,65,88,89]. For example, [39] implements a learning algorithm in the memory controller itself.

Storage overhead. The bulk of our MC design is in the storage structures that it contains. The total on-chip storage in our design is 126 KBytes. By design, the page buffers require 24 KBytes (3 pages). The other structures have empirically selected sizes: 28 KBytes for the *RemapTable* (4K entries), 64 KBytes for the cache of MQ and victim entries (4K entries), and 10 KBytes for the update and migration queues. This amount of on-chip storage is small compared to the multi-MByte shared LLC.

Our design also has limited DRAM space requirements. Taking a system with 1GB of DRAM + 32GB of PCM as a base for calculation, the total DRAM space consumed by descriptors is 6 MBytes (0.59% of the DRAM space). Each frame descriptor in the MQ queues or in the victim list takes 124 bits, which we round to 128 bits. Each descriptor contains the corresponding frame number (22 bits), the reference counter (14 bits), the queue number, including victim list (4 bits), the last-access time (27 bits), three pointers to other descriptors (54 bits), a flag indicating that the frame has been demoted (1 bit) and the counter for bad migrations (2 bits). For the configurations with which we experiment, the space taken by the descriptors is 0.63 MBytes.

3.2.2 Comparable Hybrid Memory Systems

We compare our design to the two most closely related hybrid memory systems: DBUFF [67] and WP [89].

DRAM Buffer (DBUFF) relies on a DRAM buffer logically placed between the CPU and a main memory composed solely of PCM [67]. The DRAM buffer is implemented as a set-associative cache managed entirely by the MC and invisible to the OS. Cache blocks (corresponding to virtual memory pages) coming from secondary storage are initially installed in the DRAM buffer, but also take space in PCM. From then on, the memory accesses are directed to the DRAM buffer. On a buffer miss, the page containing the desired cache line is brought into the buffer from PCM. When a block is replaced from the buffer (using the clock algorithm), it is written to its PCM frame if this is the first eviction of the block or the block was written in the buffer. Block writes to PCM are enqueued in an write buffer and done lazily in background. Like in our design, only the cache lines that were actually written are written to PCM.

When workloads exhibit good locality, most accesses hit the DRAM buffer, which leads to good performance and dynamic energy. Endurance is also good since the lazy block writes and cache-line-level writes substantially reduce the write traffic to the PCM array. (In fact, our implementation of DBUFF does not include the Fine-Grained Wear Leveling and Page-Level Bypass techniques proposed in [67]. The reason is that the endurance produced by the other techniques is sufficient in our experiments; adding extra complexity does not seem justified for our workloads and endurance assumptions.) However, workloads with poor locality may lead to poor performance and energy consumption. In addition, the inclusive DRAM caching in DBUFF reduces the amount of available memory space, potentially leading to a larger number of page faults than our design.

Our simulations of DBUFF are optimistic in many ways. First, we consider a DRAM buffer of size approximately 8% of the total memory size (rather than the original 3%). Second, we assume no DRAM buffer lookup overhead in performance or energy. Third, we implement the DRAM buffer as a fully associative structure (rather than set associative) with LRU replacement (rather than clock). Fourth, on a DRAM buffer miss requiring a page write-back, the dirty blocks (only) are written back at the same time as the missing page's content is fetched from PCM or disk.

Despite these optimistic assumptions, RaPP improves on DBUFF in two fundamental ways: (1) it uses the entire memory as a flat space, relying on page migration rather than replication; and (2) it detects when most migrations are useless and turns itself off.

Hot-modified Pages in Write Partition (WP) places DRAM and PCM in a flat address space and treats DRAM as an OS-managed write partition [89]. All pages are initially stored in PCM. The idea is to keep the cold-modified (infrequently written) pages in PCM, trying to take advantage of its low idle power consumption, and the hot-modified (frequently written) pages in DRAM to avoid PCM's high write latency and poor endurance. The MC implements a variation of the MQ algorithm with 16 LRU queues, but only counts write accesses to the physical frames. Frames that reach queue 8 (receive 2^8 writes) are considered to store hot-modified pages. On a page fault, the OS brings the page from secondary storage to the PCM area. Over time, the pages that become hot-modified are migrated to DRAM by the OS. At the same time, a page currently in DRAM but with fewer writes may have to be migrated back to PCM.

Our simulations of WP are also optimistic, as we do not charge any performance or energy overheads for the data structures and hardware modifications necessary to implement WP.

Despite these optimistic assumptions, there are three main problems with WP. First, it can hurt the performance of read-dominated workloads under less optimistic assumptions about PCM read performance. Second, migrating pages using a core at the OS quantum boundary wastes opportunities to improve performance and energy-delay within that timespan. Third, endurance also suffers because it takes a large number of writes until the OS will consider migrating a heavily written page to DRAM. Our evaluation studies mainly how WP compares to other approaches. However, in Section 3.3.3, we also isolate the impact of migrating frequently-read pages and enabling migrations within the OS quantum via hardware.

RaPP improves on WP by: (1) migrating pages that are read-intensive as well; (2) migrating pages in the background, without OS involvement; (3) including mechanisms for identifying pages worthy of migration and self-disabling for when migrations are mostly useless; and (4) spreading migrations across many physical frames. Moreover, our study improves on [89] by: (5) assuming more realistic PCM characteristics; (6) presenting a comparison of RaPP and WP to DBUFF, across a large set of workloads and parameters; and (7) evaluating other WP designs.

3.3 Evaluation

In this section, we evaluate hybrid memory systems using energy and performance as first-order metrics. Although we also report endurance results, we give them lower emphasis because our system can be easily combined with many previously proposed techniques to mitigate the PCM endurance problem (Section 5.2). Our evaluation is based on simulation, since PCM hardware is not yet available.

3.3.1 Methodology

Workloads. We simulate combinations of benchmarks from the SPEC 2000, SPEC 2006, and Stream suites forming a total of 27 workloads (Table 3.1). Because our workloads have widely different memory footprints, we group them with respect to footprint size into Large (LG), Medium (MD), and Small (SM) classes.

Simulation infrastructure. To reduce simulation times, our simulations are done in two steps. In the first step, we use M5 [9] to collect memory access (LLC misses and write-backs) traces from our workloads running on an 8-core server. Each benchmark in

Table 3.1: Workload described by tag, memory footprint in MB, LLC misses per 1000 instructions (MPKI), and percentage of LLC write-backs as a fraction of all memory accesses (WB%). Applications marked with * belong to Spec 2006.

Tag	Footprint (MB)	MPKI	WB%	Applications (x2 each)
LG1	993	12	33	milc*, gobmk*, sjeng*, libquantum*
LG2	992	29	32	S.add, S.copy, apsi, milc*
LG3	746	24	27	mcf*, S.triad, sjeng*, facerec
LG4	743	4	25	vortex, milc*, sixtrack, mesa
LG5	702	24	26	sjeng*, S.triad, S.add, swim
LG6	683	4	28	perlbmk, crafty, gzip, milc*
LG7	645	25	32	lucas, gcc, mcf*, sphinx3*
LG8	594	18	32	wupwise, vpr, mcf*, parser
LG9	557	17	32	swim, eon, art, lucas
MD1	486	13	49	applu, lucas, gap, apsi
MD2	467	23	32	S.scale, S.triad, swim, eon
MD3	414	20	30	mcf*, parser, twolf, facerec
MD4	407	8	24	namd*, S.triad, sjeng*, wupwise
MD5	394	13	32	art, lucas, mgrid, fma3d
MD6	385	24	28	art, mcf*, gzip, vpr
MD7	381	14	23	S.add, h264ref*, equake, hmmer*
MD8	367	46	33	S.triad, S.add, S.copy, S.scale
MD9	356	30	27	equake, S.scale, S.triad, mgrid
SM1	295	2	21	wupwise, gobmk*, vortex, h264ref*
SM2	285	5	33	swim, perlbmk, namd*, eon
SM3	283	6	33	swim, crafty, twolf, gcc
SM4	276	16	33	lucas, h264ref*, libquantum*, sphinx3*
SM5	271	15	27	wupwise, equake, ammp, libquantum*
SM6	260	11	24	fma3d, mgrid, galgel, equake
SM7	247	12	32	fma3d, sphinx3*, galgel, lucas
SM8	243	15	21	S.triad, h264ref*, fma3d, equake
SM9	243	2	29	ammp, gap, wupwise, vpr

Table 3.2: System settings.

Feature		Value	
CPU cores (2.668GHz, Alpha ISA)		8 in-order, one thread/core	
L1 I/D cache (per core)		64KB, 2-way, 1 CPU cycle hit	
L2 cache (shared)		8MB, 8-way, 10 CPU cycle hit	
Cache block size / OS page size		64 bytes / 8KB	
Memory (667MHz/DDR3-1333)		8KB rows, close-page	
Memory devices (x8 width, 1.5V)		DRAM	PCM
Delay	tRCD	15ns	56ns
	tRP	15ns	150ns
	tRRDact	6ns	5ns
	tRRDpre	6ns	27ns
	Refresh time	64ms	n/a
	tRFC / tREFI	110ns / 7.8μs	n/a
Current	Row Buffer Read	200mA	200mA
	Row Buffer Write	220mA	220mA
	Avg Array R/W	110mA	242mA
	Active Standby	62mA	62mA
	Precharge Powerdown	40mA	40mA
	Refresh	240mA	n/a
Normalized Density		1	4
Data Retention		64 ms	> 10 years
Cell endurance (writes)		$> 10^{16}$	$10^8 - 10^9$

a workload is represented by its best 100M-instruction simulation point (selected using Simpoints 3.0 [66]). A workload terminates when the slowest application has executed 100M instructions.

In the second step, we replay the traces using our own detailed memory system simulator. This simulator models all the relevant aspects of the OS, memory controller, and memory devices, including inverted page tables, contention, memory device power and timing, and row buffer management. Our decision to develop our simulator stemmed from our need, at the time, for a tool that simulated DDR3 in detail, in reasonable time, and that was simple to expand.

The main architectural characteristics of the simulated server are listed in Table 3.2. We simulate in-order cores to expose the overheads associated with PCM accesses to workloads. The cores have private 64-Kbyte 2-way instruction and data L1 caches, as well as an 8-MByte 8-way combined shared cache. For this cache architecture, Table 3.1

reports the LLC misses per kilo instruction (MPKI) and the percentage of LLC writebacks (WB%) for each workload. The memory system has 4 DDR3 channels, each one occupied by a single-rank DIMM with 8 devices (x8 width) and 8 banks per device. In all simulations, we assume an initially warm memory (no cold page faults). The MC implements cache-block-level bank interleaving and page-level channel interleaving. Memory accesses are served on a FCFS basis. The MC uses close-page row buffer management. (More sophisticated access scheduling is not necessary for our simulated system and workloads, as opportunities to increase their bank hit rate via scheduling are rare, and such improvements are orthogonal to our study.)

A memory rank can be in (1) Active Standby state, when at least one of its banks is serving requests; or (2) Precharge Power Down, when all banks are idle and the clock enable line is turned off to save energy. Additionally, PCM is enhanced to avoid writing unmodified cache lines back to the cell array. The table shows power parameters [47] of DRAM and PCM chips, and the timing parameters that change across memory technologies [47, 67, 89]. The timing parameters are: the time between the issue of a row activation command and the data being accessible at the row buffer of a given bank (tRCD); the time between two row activation commands issued to the same rank after the evicted row was read (tRRDact) or written (tRRDpre); the time between a precharge and an activation to a same rank (tRP); the time between two refresh commands or a refresh and an activation (tRFC); the maximum time between two refresh commands for a same row (Refresh time); and the average delay between refresh commands across all rows (tREFI). In addition to these parameters, we simulate 12 other timing constraints that are relevant for our study [55, 83].

Besides RaPP, we simulate the two hybrid approaches mentioned in Section 3.2 (DBUFF and WP) and an additional "Unmanaged" system, in which pages remain in the frames originally assigned to them by the OS. We use Unmanaged as the baseline for comparison. *Only RaPP is assumed to have energy and performance overheads stemming from its data structures.* Our assumptions for RaPP are consistent with those of other authors [30]. For example, the RaPP SRAM consumes 0.13W of background power and 0.017nJ per 16-byte operation; the transfer buffers consume

0.06W of background power and 0.06nJ per 64-byte operation; and each DIMM-level row buffer increases the rank background power by 12.5% when active (and as much dynamic power as a regular row buffer). The four hybrid systems have 1 channel equipped with 1 DRAM DIMM (128MB) and the remaining 3 channels with 1 PCM DIMM each (3x128x4MB=1536MB), totaling 1.664 GBytes of memory. We picked these small memory sizes to match the footprint of the workloads' simulation points.

As another basis for comparison, we use a PCM-only system with 2 GBytes of memory (4x128x4MB). Previous works have shown that the DRAM-only system exhibits much worse performance than PCM-only and hybrid systems [67], due to its lower storage capacity, so we do not consider it in this chapter.

3.3.2 Results

3.3.2.1 Performance and energy

We now compare the behavior of RaPP, DBUFF, WP, and the two baseline systems. Due to space limitations, we do not present separate performance and energy results for all workloads; instead, we plot these results for the MD workloads and discuss the results for other workloads in the absence of figures. Later, we plot energy-delay^2 (ED2) results for all workloads.

Figure 3.4 presents the running time (top graph, including the geometric mean of the MD results), average memory power consumption (middle graph, including the power consumed by the SRAM cache used in RaPP), and number of page migrations in RaPP and WP and page fetches from PCM in DBUFF (bottom graph, again including the geometric mean). We refer to page migrations and fetches collectively as page transfers. The performance and power results are normalized to Unmanaged. Note that we plot average power, rather than energy, to remove the impact of different running times (which are plotted in the top graph).

Running time. The top graph shows that RaPP exhibits the lowest average running times, followed by WP, Unmanaged, PCM-only, and then DBUFF. In fact, RaPP performs better than PCM-only and Unmanaged for all workloads (including the LG and SM workloads). RaPP achieves these results by migrating popular pages to the DRAM area, which has substantially better performance in the presence of row buffer misses than PCM. WP and DBUFF do not always outperform PCM-only and Unmanaged. RaPP is more robust than WP and DBUFF, achieving good performance in most cases and preventing degradation in others by disabling itself.

WP attempts to migrate pages to DRAM as well, but focuses solely on those pages that experience a large number of write-backs (or writes of disk data read into memory). In addition, the fact that WP migrates pages at the end of the OS quantum has two sides. On the negative side, WP misses opportunities to migrate popular pages as their popularity shifts within the OS quantum. MD2 and MD8, for example, suffer from this problem. On the positive side, migrating infrequently reduces the number of migrations in workloads where most migrations will turn out useless. However, RaPP is more effective than WP at preventing unnecessary migrations, as it includes mechanisms designed explicitly for this purpose. For example, in SM2, a large fraction of pages is hot-modified, but performance-irrelevant. In that case, RaPP disables itself at about a quarter of the execution. A similar phenomenon occurs in 6 other workloads in our experiments. In essence, RaPP is more effective at migrating the actual performance-critical pages to DRAM, improving performance by 6% for MD workloads and 7% overall with respect to WP.

The most extreme results happen for DBUFF. It achieves the lowest running time for MD1, but dismal running times for MD3 and MD6. The reason for DBUFF's poor performance is that MD3 and MD6 exhibit poor locality (their working sets are substantially larger than the size of the DRAM buffer). The same effect occurs for several LG workloads. Poor locality forces frequent page fetching from PCM into DRAM, with the eviction of dirty blocks (if any) within victim pages done in the background. Without the problematic workloads, RaPP still performs 14% and 8% better than DBUFF for the LG and MD workloads, respectively. On the other hand, the SM workloads have working sets that easily fit in the DRAM buffer, so DBUFF performs best for 6 of them. However, RaPP's overall average performance is still slightly better than DBUFF's for the SM workloads.

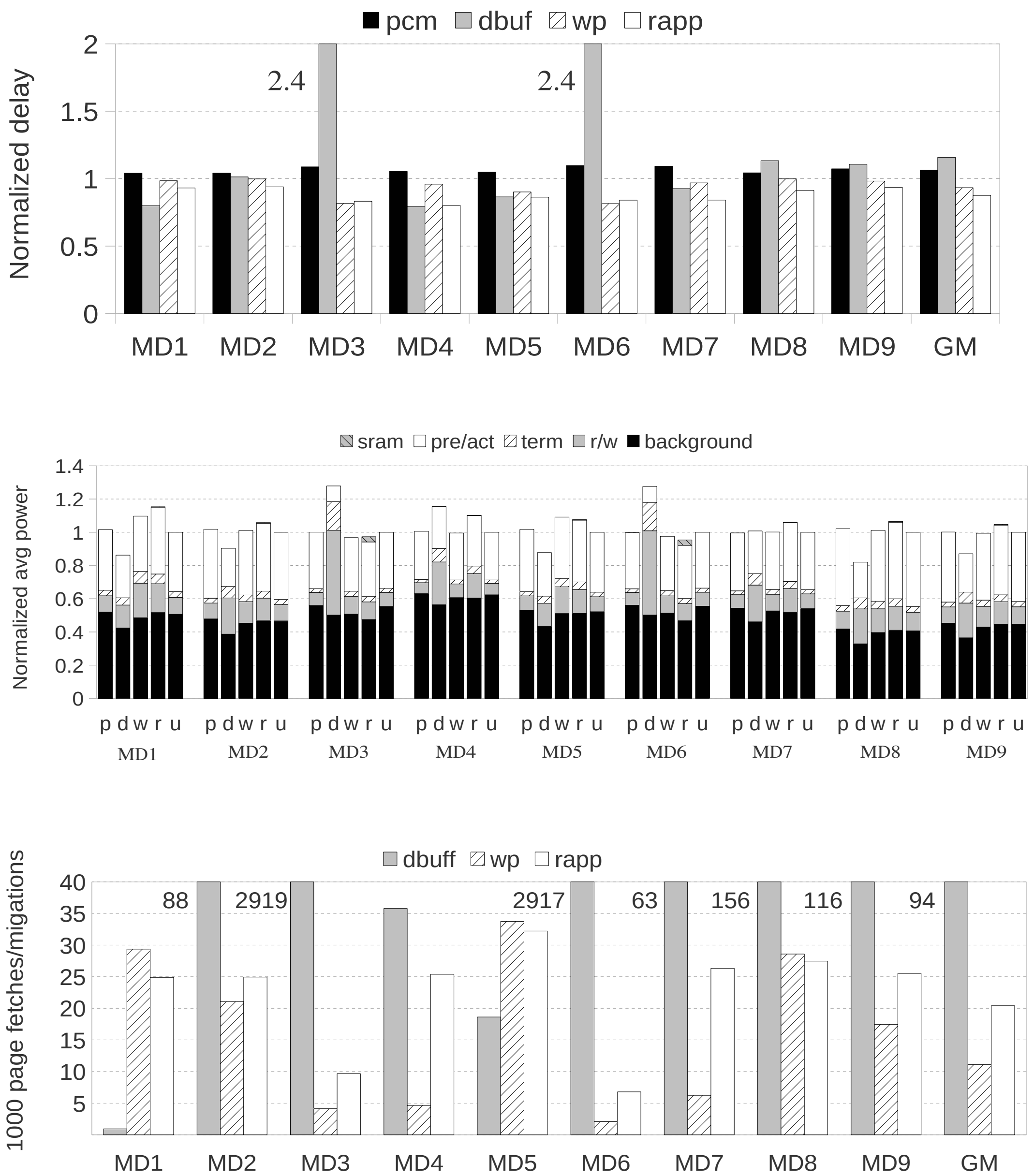


Figure 3.4: Comparing performance, average power, and page transfers for the MD workloads. In the middle figure, p = PCM-only, d = DBUFF, w = WP, r = RaPP and u = Unmanaged.

Finally, note that Unmanaged consistently outperforms PCM-only. The reason is that Unmanaged benefits from accessing data in its DRAM section, which is substantially faster than accessing data in PCM when row buffer misses occur. (Excluding the effect of page transfers, the row buffer miss ratio we observe is always higher than 80%. This effect has been observed in previous studies of multi-core servers as well, e.g. [77].)

Average memory power. Considering the middle graph of Figure 3.4, we see that all systems exhibit similar average power consumption for the MD workloads (the same happens for the LG and SM workloads). The average power correlates well with the total number of accesses in each approach. As expected, page transfers increase read/write and activation/precharge (activation/write to array for PCM) power somewhat. We also see that DBUFF produces lower background power. The reason is that the PCM area can be in precharge powerdown state more often in DBUFF.

Interestingly, although RaPP uses an SRAM cache for its ranking of pages, the power consumed by this cache is negligible. Most rank accesses become cache hits (at least 90% in our experiments). The SRAM results in the figure account for the static and hit energies. The energy consumed by the misses is reported in the other categories.

Page transfers. The bottom graph shows that RaPP migrates more pages than WP for most (all but 3) MD workloads. The same happens for most SM (all but 1) and LG (all but 2) workloads. The reason is that each migration operation in RaPP actually transfers 3 pages, instead of 1 or 2 pages in WP. Interestingly, DBUFF fetches many more pages from PCM than RaPP and WP migrate in the MD workloads. Again, the reason is that these (and the LG) workloads have working sets that do not fit in the DRAM buffer. For the SM workloads, DBUFF fetches pages much less frequently than for the MD and LG workloads. In fact, DBUFF transfers fewer pages than WP in 4 SM workloads and fewer than RaPP in 6 of them.

Putting performance and energy together: ED2. Figure 3.5 plots the ED2 of each workload and system normalized to Unmanaged. The rightmost set of bars in each graph shows the geometric mean of the results presented in the same graph. The graphs

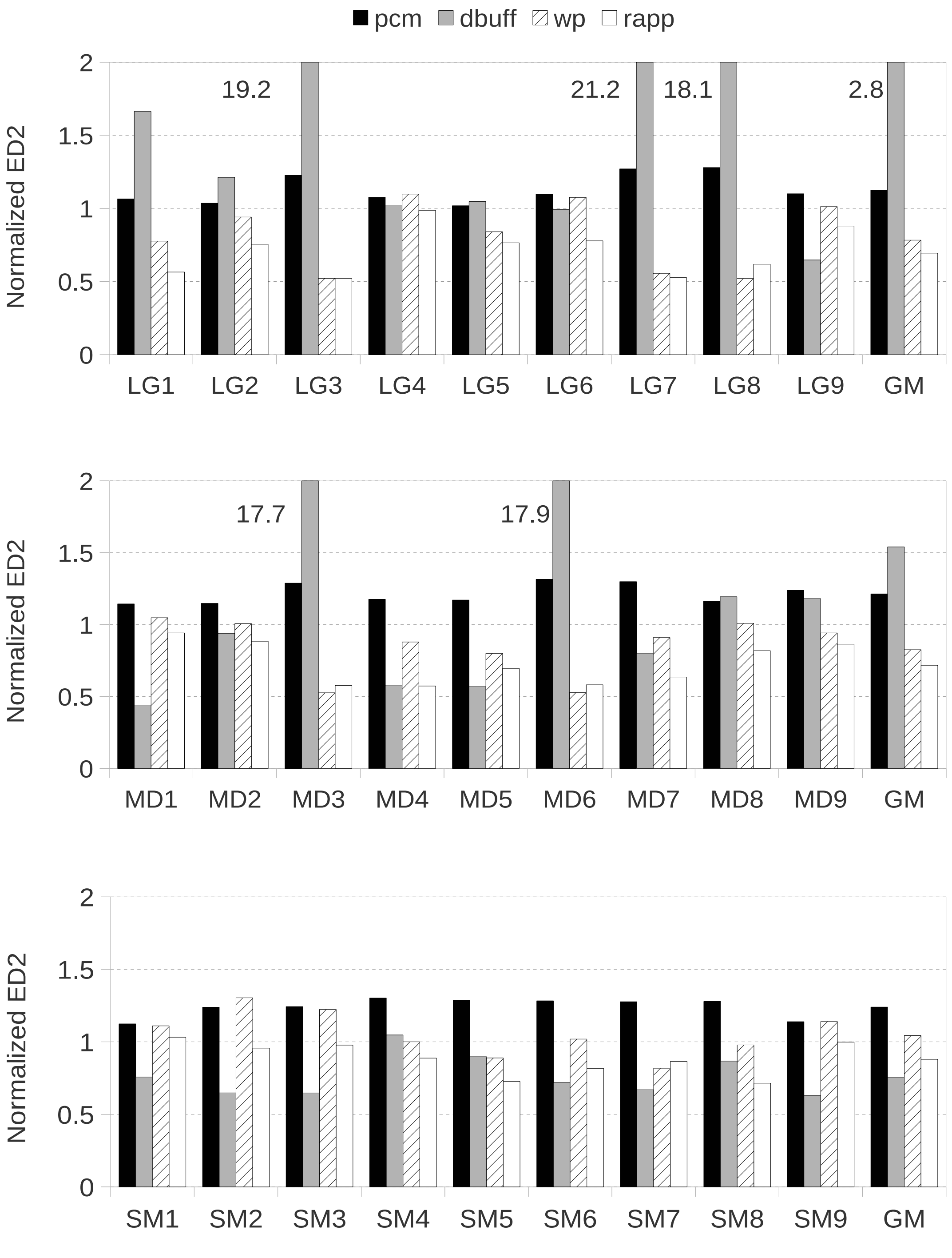


Figure 3.5: Comparing energy-delay^2 for all workloads.

show that RaPP is the only system to achieve ED2 no higher than Unmanaged and PCM-only for all workloads. (In the few instances where RaPP achieves roughly the same ED2 as Unmanaged, it disabled itself due to a large percentage of bad migrations.) Considering the workload classes independently, we find that RaPP achieves the lowest average ED2 for the LG and MD workloads. For the SM workloads, the combination of low run time, very few migrations, and idle PCM ranks (as discussed before) leads DBUFF to the lowest average ED2 (14% lower than RaPP). Overall, for the workloads we study, RaPP achieves 13%, 24%, 36%, and 49% lower ED2 than WP, Unmanaged, PCM-only, and DBUFF, respectively.

As mentioned above, RaPP improves ED2 over PCM-only and Unmanaged by migrating popular pages from PCM to DRAM. The comparison to WP is more interesting. RaPP and WP achieve comparable ED2 for some workloads. However, WP achieves worse ED2 than Unmanaged for 2 LG, 1 MD, and 4 SM workloads. In many of these cases, RaPP did very well but in others it simply disabled itself. Compared to DBUFF, RaPP wins for workloads with working sets larger than the DRAM buffer (i.e., 9 of our workloads).

3.3.2.2 Endurance

To evaluate the system lifetimes (limited by PCM's endurance), we resort to the Required Endurance metric [11]: $T_{life} \times \frac{B}{\alpha\ C}$, where B is the memory bandwidth in bytes per second, T_{life} is the desired system lifetime in seconds, α is the wear-leveling efficiency of the system, and C is the memory capacity in bytes. For each workload, we consider T_{life} the typical 3- to 5-year server lifespan. α is A/M, where A is the average number of writes per PCM cache block and M is the number of writes to the most written cache block in the PCM area of the system. A low value of α suggests poor wear-leveling. Required Endurance determines the number of writes that a PCM cache block must be able to withstand during the server's lifespan.

Figure 3.6 compares the base-10 logarithm of the Required Endurance of the systems and workloads we consider, using the 3- and the 5-year projections. For comparison, PCM's endurance today is $10^8 - 10^9$. The figure shows that RaPP requires roughly 10%

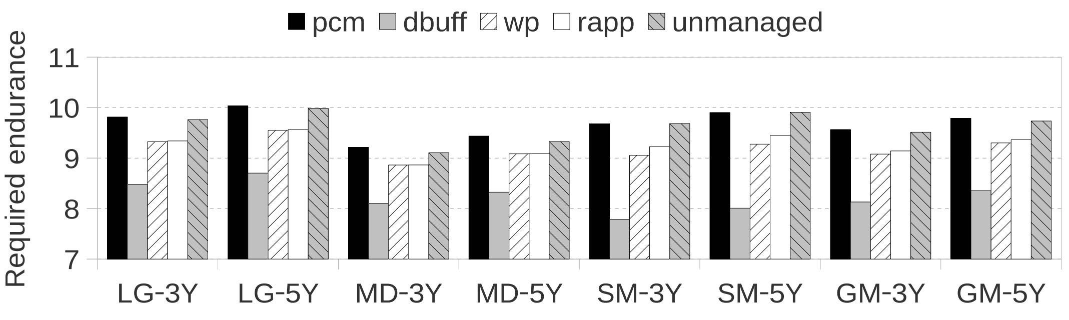


Figure 3.6: Required Endurance projected over 3 and 5 years.

more endurance than DBUFF, but only 1% more than WP on average. In contrast, it requires 5% and 4% less endurance than PCM-only and Unmanaged, respectively. DBUFF provides the best Required Endurance.

In RaPP's 5-year projection, most MD and SM workloads require endurance only slightly higher than 10^9, whereas a few LG workloads require closer to 10^{10} endurance. The reason for these results is that in RaPP the frequently read and frequently written pages compete for space in the DRAM area, trying to strike a balance between performance and endurance. In addition, in some cases, RaPP's self-disabling mechanism is triggered, making it behave as Unmanaged from that point on.

However, we argue that RaPP's endurance results are not a cause of concern for two reasons. First, as we mention in Section 5.2, many proposals already extend the lifetime of PCM greatly. Several of them are orthogonal to RaPP [16, 40, 47, 90] and can enhance PCM independently from our approach. Second, the predictions for future PCM cells suggest significant endurance improvement compared to DRAM (10^{12} writes by 2012 [17] and 10^{15} writes by 2022 [36]), but performance and energy will still lag. Better PCM endurance justifies a shift towards reducing energy and delay, as we do in this dissertation.

3.3.3 Sensitivity analysis

RaPP parameters. We now study the impact of the migration threshold (MT), which defines the number of accesses to a PCM page before deciding to migrate it to DRAM; the filter threshold (FT), which defines the window of time for filtering consecutive

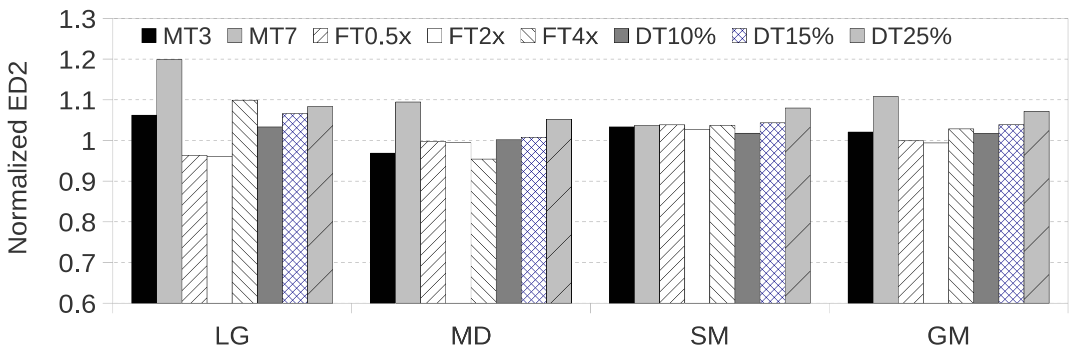


Figure 3.7: RaPP's sensitivity to migration threshold (MT), filter threshold (FT), disable threshold (DT).

page references; and the disable threshold (DT), which defines the percentage of bad migrations before RaPP disables itself.

Figure 3.7 depicts the ED2 results for the three workload classes and the overall geometric mean. Each bar is normalized to the ED2 resulting from the default value of the corresponding parameter. Specifically, the default value for MT is 2^5 and we compare it to 2^3 (labeled MT3) and 2^7 (MT7); the default value for FT is $MigrationTime/32$ and we compare it to 0.5x default FT (FT0.5x), 2x default FT (FT2x), and 4x default FT (FT4x); and the default value for DT is 5% and we compare it to 10% (DT10%), 15% (DT15%), and 25% (DT25%). We always vary one parameter at a time, keeping others at their default values.

The figure shows that RaPP is most sensitive to MT, especially for the LG workloads. In particular, MT7 degrades ED2 by 20% on average for the LG workloads, compared to the default. It also degrades ED2 for the MD and SM workloads. In contrast, MT3 degrades ED2 for the LG and SM workloads, but not by as much and not for the MD workloads. RaPP exhibits relatively low sensitivity to FT, except for FT4x for LG workloads. Finally, RaPP consistently showed more ED2 degradation as DT increases.

These results suggest that (1) overshooting the ideal MT is more harmful on average than undershooting it; and (2) lower values for FT and DT provide better behavior. In contrast, varying these settings has a negligible impact on Required Endurance; it varies only within 1% compared to our default RaPP results.

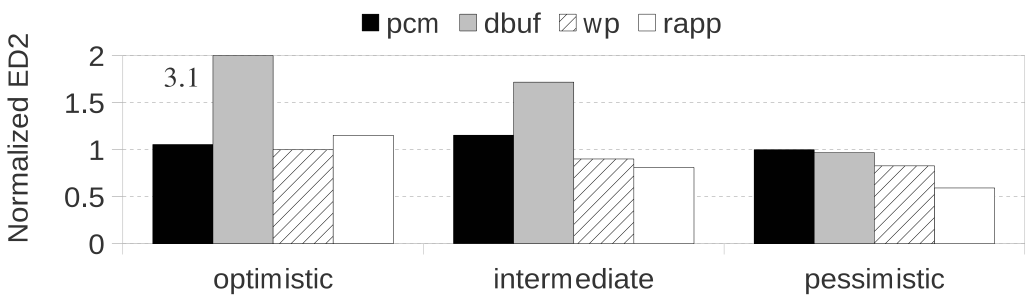


Figure 3.8: Sensitivity to PCM's characteristics.

PCM's characteristics. We now evaluate the impact of the performance and energy characteristics of PCM devices. We consider three additional settings for PCM performance: optimistic, intermediate, and pessimistic; the PCM energy varies along with its performance settings. Specifically, the optimistic setting assumes that row activations are 40% faster than our default value, bringing them close to DRAM's activations; the array writes stay with their default values. [89] assumed similarly high-performing PCM activations. The pessimistic setting assumes that activations and array writes are 25% and 2.5x slower than our defaults, respectively. [21] assumed similarly pessimistic parameters. Finally, the intermediate setting assumes that activations are 25% faster and array writes are 50% slower than our defaults.

Figure 3.8 shows the average ED2 results (across all workloads) normalized to Unmanaged. We can see that RaPP achieves the best average ED2 for the pessimistic and intermediate cases. The advantage of RaPP is particularly significant in the pessimistic scenario. For the optimistic setting, PCM-only and WP achieve roughly the same ED2 as Unmanaged. This is not surprising since the optimistic performance and energy of PCM become comparable to those of DRAM. Because RaPP attempts to migrate pages to DRAM despite the optimistic assumptions for PCM, it achieves slightly higher ED2. In contrast, DBUFF achieves worse ED2 because of the workloads with poor locality.

These results suggest that RaPP behaves better than the other systems for different sets of expected PCM characteristics (default and intermediate). RaPP's benefits will be even more pronounced, if commercial PCM devices turn out to be worse than our

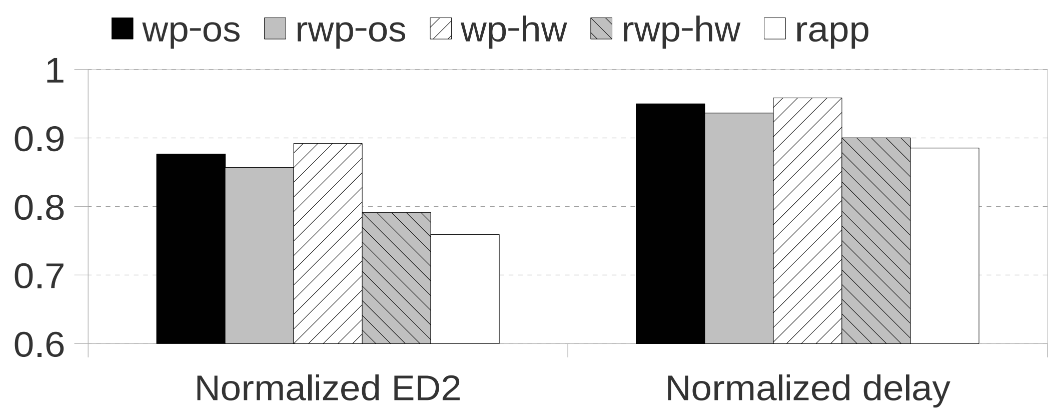


Figure 3.9: Comparing RaPP to different flavors of WP.

expectations. Again, the Required Endurance varies negligibly (less than 2%) across these parameter settings.

WP variants and RaPP. Here, we isolate the effect of two important differences between RaPP and WP: (1) the ability to migrate hot pages in general (vs. only hot-modified ones) and (2) the ability to migrate pages in hardware within the OS quantum.

In addition to the original WP (WP-OS), we study three variants. The first variant (RWP-OS) migrates frequently read pages, as well as hot-modified ones at the end of the OS quantum. The second variant (WP-HW) is hardware-driven, enabling migrations within the OS quantum without any overhead (e.g., a processor interrupt). The last variant (RWP-HW) is hardware-driven and migrates frequently read and written pages. In all versions of WP, we migrate a PCM page that reaches the best migration threshold in our experiments. (We determined the best threshold by running simulations with every possible threshold and computing their corresponding average ED2 results.) Specifically, we migrate the PCM pages that reach queue 3 for WP-OS and WP-HW, and queue 5 for RWP-OS and RWP-HW. We also disconsider *all* MQ-related overheads, which would be present in all four WP versions. These assumptions are far more optimistic than our assumptions for RaPP.

Figure 3.9 shows the average ED2 and delay results (across all workloads) normalized to Unmanaged. The figure shows that migrating frequently read pages at the end of the

OS quantum (RWP-OS) only improves average performance by 1% and average ED2 by 2% over WP-OS. However, we observe that for most workloads, page popularity shifts fast within the OS quantum, causing RWP-OS (and WP-OS) to overlook opportunities for migration, as we discuss next. Interestingly, RWP-OS hurts endurance in most workloads as compared to WP-OS, because read-critical pages compete with hot-modified ones, delaying or detracting the system from migrating endurance-critical pages in many workloads and/or from retrying to migrate them until the next OS quantum. Lowering the migration threshold to force causes DRAM thrashing, worsening the average performance and ED2.

With sub-quantum migrations, WP-HW moves more endurance-critical pages into DRAM than WP-OS, improving endurance by 1% on average. However, it worsens performance by 1% and ED2 by 2%, due to an increased fraction of bad migrations. Increasing WP-HW's migration threshold to reduce the number of migrations worsens performance and ED2 even further.

RWP-HW achieves the best average improvement over WP-OS (5% in performance and 10% in ED2). This result is mainly due to the fact that migrations occur while a page is gaining popularity. It is also possible to retire accesses in the transfer buffers within the MC itself and avoid accesses to RAM altogether, which is not possible in the OS approach. However, without self-disabling, RWP-HW exhibits ED2 up to 33% worse than Unmanaged.

3.4 Summary

In this chapter, we introduced a novel hybrid memory system design combining DRAM and Phase-Change Memory. Our design features a sophisticated memory controller, and a page ranking and migration policy called RaPP. The memory controller monitors the memory accesses and implements RaPP. RaPP seeks to benefit from the best characteristics of DRAM and PCM, while avoiding their limitations. The policy takes into account the recency and frequency of page accesses, as well as the write traffic to each page, to rank pages and lay them out in physical memory. Our results demonstrate that for the workloads we study, our design behaves significantly better than two state-of-the-art hybrid designs, despite our optimistic assumptions for the latter systems.

Chapter 4

Rank-aware Cooperative Cache

4.1 Overview

So far, hybrid memory approaches combining DRAM and PCM are either not robust to some workloads and/or require hardware modifications. Moreover, hybrid memory systems have not yet been considered for server clusters, such as those of modern Internet services.

Interestingly, Internet services are a great target for hybrid memory systems, since these services require low latency and high memory capacity at the same time. In fact, these services organize the entire set of server main memories into a single, cluster-wide cooperative cache. By doing so, they can avoid accessing slow disks or re-generating content. For example, Facebook, Twitter, Youtube, and Wikipedia are known to use the Memcached cooperative caching middleware [26]. Other proposals, such as the PRESS middleware [12], also implement cooperative caches, but in a more flexible fashion than Memcached.

In this chapter, we propose RaCC, a software-driven object placement policy for hybrid memory systems in server clusters that implement cooperative caches. RaCC monitors object popularity and leverages that information in placing the objects across servers and memory technologies. Specifically, RaCC concentrates popular cached objects in the collective DRAM of the server cluster, while taking advantage of PCM's large capacity to increase the hit ratio of the cooperative cache. We apply RaCC to both Memcached and PRESS, using either solid-state drives (SSDs) or hard disk drives (HDD) for persistent storage.

We evaluate these RaCC-based systems using simulation of real and synthetic traces of Internet services. For comparison, we also simulate Memcached and PRESS with DRAM only, PCM only, and a hybrid memory system without RaCC (called "unmanaged hybrid" or simply "unmanaged"). The unmanaged system does not distinguish between DRAM and PCM. Our results show that RaCC adds performance robustness to the hybrid cooperative caches. Considering the workloads we study, for PRESS, RaCC improves the average performance per request by 43%, 30%, and 25% respectively, compared to the DRAM-only, PCM-only, and unmanaged systems. For Memcached, RaCC's performance improvements are respectively 42%, 32%, and 20%. We observe that RaCC's gains depend on the workload locality and the performance of the persistent storage device. To achieve robustness, RaCC systems do not consume more energy than the other clusters. PCM endurance is not a problem for any of the systems we consider.

4.2 Cooperative caching in Internet services

To meet their performance requirements, modern Internet services aggressively cache Web objects. In fact, they often use a middleware layer that creates a large cluster-wide cooperative cache to which each server contributes some amount of main memory [12,18,26,64]. The middleware then directs each Web object request to the server likely to be caching the object. In this chapter, we study two such middlewares: Memcached and PRESS.

4.2.1 Memcached

Memcached [26] relies on object id/key hashing for object placement and load balancing. In more detail, Memcached hashes each object request to one server and sends the request to it. If the server caches the object, it simply replies with the object. Otherwise, it fetches the object from secondary storage, caches it locally, and replies to the request with the object. In this case, Memcached replaces objects locally to create space for the new object using LRU replacement. For maximum scalability, servers do not communicate in processing requests. However, Memcached does not replicate objects, which may cause load imbalance. To avoiding thrashing the cache with large but unpopular objects, Memcached limits the size of the largest cacheable object.

4.2.2 PRESS

In contrast, PRESS [12] does explicit locality-aware load distribution, and distributed load balancing via request forwarding and object replication. PRESS is based on the observation that accessing a remote memory is faster than accessing local storage. This observation holds even when servers use SSDs but are interconnected with high-performance switches (e.g., InfiniBand).

PRESS implements request forwarding and load balancing with the help of a Global Directory (GD) structure, replicated in every server. Each server updates the GD in two situations: (1) whenever it starts or stops caching an object; and (2) periodically, to inform the other servers about its load level (implied from the number of open connections). In PRESS, any server can receive requests directly from clients (e.g., via round-robin DNS). Upon receiving a request, a server consults the GD and decides whether to (1) forward the request to a non-overloaded server that currently caches the object; (2) cache the object locally if there is no other (non-overloaded) server caching the object; or (3) request the least loaded server in the cluster to cache the object, if the servers that currently cache it and the receiving server are overloaded. Each server manages its local memory using LRU replacement. As in Memcached, PRESS limits the size of the largest cacheable object. We show the pseudo-code for PRESS' request distribution in Algorithm 1.

4.3 The RaCC Policy

In this section, we present the RaCC policy. RaCC is an object placement policy for cooperative caches where the memory of each server comprises both DRAM and PCM. Given the characteristics of DRAM and PCM, RaCC seeks to improve response time by caching popular objects in DRAM and ultimately serving as many requests from that memory region as possible. Previous research motivated our work by showing that

```
function coopCacheServe (Request r)
begin
    Object obj = r.target
    if obj.size > MAX_SIZE then
        replyToClient (r.client, serveFromDisk (r))
    end
    else if obj is not cached on any server [†] then
        startCaching (obj)
        replyToClient (r.client, serveFromCache (obj))
    end
    else if obj is cached on the initial server then
        replyToClient (r.client, serveFromCache (obj))
    end
    else
        balanceLoad (r, obj)
    end
end
function balanceLoad (Request r, Object obj)
begin
    PeerServer w = least loaded server caching obj[†]
    if GD.notOverloaded (w) then
        forwardToPeer (r, w)
        replyToClient (r.client, receiveReplyFromPeer (w))
        return
    end
    if GD.notOverloaded (localhost) then
        startCaching (obj)
        replyToClient (r.client, serveFromCache(obj))
        return
    end
    PeerServer v = least loaded server in the cluster[†]
    if GD.notOverloaded (v) then
        forwardToPeer (r, v)
        replyToClient (r.client, receiveReplyFromPeer(v))
    else
        forwardToPeer (r, w)
        replyToClient (r.client, receiveReplyFromPeer(w))
    end
end
function receiveForwardedRequest (Request r, PeerServer p)
begin
    if r.target is not cached then startCaching (r.target)
    replyToPeer (p, serveFromCache (r.target))
end
```

Algorithm 1: Request service in PRESS ([†]via GD)

typically only a relatively small subset of Web objects are very frequently accessed in Web caches [29, 76]. Moreover, that subset may change over time. These observations suggests that (1) the most popular objects may fit entirely in the collective DRAM area of the cooperative cache; (2) the majority of lightly accessed objects will consume lower energy if accessed in PCM rather than HDD or SSD; and (3) the system must dynamically identify the popular objects and adjust their placements accordingly.

We first describe RaCC's basic mechanisms, then we explain how the approach can be used on PRESS and Memcached.

4.3.1 Basic mechanisms

Object ranking. To make object placement decisions, RaCC tracks the popularity of objects stored in the cooperative cache and leverages that information in placing them. Specifically, RaCC tracks the frequency and recency of references to the cached objects. To dynamically rank objects, each server individually builds two private MQ structures [91], updated as requests for content arrive at the servers. The first MQ structure (*PMQ*) ranks objects that are stored solely in PCM, whereas the second structure (*DMQ*) ranks objects that are stored solely in DRAM. RaCC does not allow objects to span the two memory regions. Each MQ structure is similar to the original structure described in Section 3.2.1, with the difference that here we rank objects rather than frames.

When an object cached in PCM reaches queue *PopThres* of PMQ, it is considered popular. In our simulations, we adopt 16 queues per MQ structure, and we set *LifeTime* to 32 and *PopThres* to 8, because they showed good empirical performance and energy results.

Server-level object placement. RaCC manages memory as shown in Algorithm 2. RaCC tries to allocate objects first in DRAM, then in PCM, if they have free frames available. The function *cacheObject* maps the newly cached object to free virtual pages and copies the object's content from the storage device into a corresponding set of physical memory pages. When neither memory region has free space, the replacement

```
1  function startCaching (Object obj)
2  begin
3      if DRAM.hasEnoughFreeFrames(obj.size) then
4          cacheObject (obj, DRAM)
5          DMQ.add (obj)
6      end
7      else if PCM.hasEnoughFreeFrames(obj.size) then
8          cacheObject (obj, PCM)
9          PMQ.add (obj)
10     end
11     else
12         DMQpivot = DMQ.getLeastPopularInSet(obj.size)
13         PMQpivot = PMQ.getLeastPopularInSet(obj.size)
14         if DMQpivot is less popular than PMQpivot then
15             ensureRoom (obj.size, DRAM, DMQ)
16             cacheObject (obj, DRAM)
17             DMQ.add (obj)
18         end
19         else
20             ensureRoom (obj.size, PCM, PMQ)
21             cacheObject (obj, PCM)
22             PMQ.add (obj)
23         end
24     end
25 end
```

Algorithm 2: RaCC's local placement algorithm

algorithm coordinates DMQ and PMQ, seeking to replace the least popular content in the entire memory. Because RaCC manages DRAM and PCM disjointedly, the replacement algorithm must select one of these memory regions, then make room in it for the new object. To select the region, RaCC finds the set of least popular objects in DMQ and PMQ (lines 8–9), then compares the popularity of a "pivot object" (the most popular object) in each set (line 10). If the pivots occupy different MQ queues in their respective structures, RaCC selects the region with the lowest ranked pivot. Otherwise, RaCC selects the region with the LRU pivot. The function *ensureRoom* frees as many bottom-ranked objects of the selected region as necessary to fit the new object, and updates the corresponding structure. Finally, RaCC caches the new object as explained before.

```
function coopCacheServe (Request r)
begin
    Object obj = r.target
    if obj.size > MAX_SIZE then
        replyToClient (r.client, serveFromDisk (r))
    end
    else if obj is not cached on any server † then
        startCaching (obj)
        replyToClient (r.client, serveFromCache (obj))
    end
    else if obj is cached on the initial server then
        replyToClient (r.client, serveFromCache (obj))
        tryMigrate (obj)
    end
    else
        balanceLoad (r, obj)
    end
end
function balanceLoad (Request r, Object obj)
begin
    PeerServer w = least loaded server caching obj in DRAM†
    if GD.notOverloaded (w) then
        forwardToPeer (r, w)
        replyToClient (r.client, receiveReplyFromPeer (w))
        return
    end
    PeerServer z = least loaded server caching obj in PCM†
    if GD.notOverloaded (z) then
        forwardToPeer (r, z)
        replyToClient (r.client, receiveReplyFromPeer(z))
        return
    end
    if GD.notOverloaded (localhost) then
        startCaching (obj)
        replyToClient (r.client, serveFromCache(obj))
        return
    end
    PeerServer v = least loaded server in the cluster†
    if GD.notOverloaded (v) then
        forwardToPeer (r, v)
        replyToClient (r.client, receiveReplyFromPeer(v))
    else
        forwardToPeer (r, w)
        replyToClient (r.client, receiveReplyFromPeer(w))
    end
end
function receiveForwardedRequest (Request r, PeerServer p)
begin
    if r.target is not cached then  startCaching (r.target)
    replyToPeer (p, serveFromCache (r.target))
    tryMigrate (obj)
end
```

Algorithm 3: RaCC request service for PRESS (†via GD)

4.3.2 RaCC for PRESS

RaCC leverages the request distribution and object replication algorithms of PRESS to place objects and replicas in the cooperative cache, as we explain below.

Request service. Algorithm 3 depicts the PRESS behavior upon receiving a request, when augmented with RaCC. Compared to the original PRESS design, the main modifications introduced by RaCC are (1) the search for opportunities to migrate popular objects in DRAM (lines 11 and 41); and (2) the forwarding of requests preferably to servers already caching the requested object in DRAM (lines 16–25).

As we mention in Section 4.2, HTTP requests arrive via standard round-robin DNS to any server. The server that initially receives the request (called "initial server") parses it, inspects its content, and decides where it should be served. The initial server itself serves requests for objects that are (1) larger than the maximum cacheable size, which is 1MByte in our simulations (lines 4–5); (2) seen for the first time (lines 6–8); or (3) already cached locally in DRAM or PCM (lines 9–10). Otherwise, the initial server tries to forward the request to another server while balancing load (lines 12–13). Using the GD, PRESS finds the least loaded server that already has the object in memory. If that server is not overloaded, PRESS forwards the request to it. RaCC breaks this step into two. First, it looks for a server that has the object in DRAM (lines 16–19). If that server is not found, RaCC looks for a server caching the object in PCM (lines 21–24). If none of those are found, the initial server tries to cache the object in its own memory, as long as it is not overloaded (lines 26–29). Otherwise, again using the GD, the initial server looks for the least loaded server in the cluster. If that server is not overloaded, the initial server forwards the request to it (lines 30–33). If none of the previous cases is satisfied, RaCC forwards the request to the server that has the object in DRAM (lines 35–36). Upon receiving a forwarded request, a server may need to update its cache, and then send the cached object to the initial server (lines 37–41). Whenever a server starts caching the requested object, it broadcasts that fact to all servers, updating the GD, as in the original PRESS design. In that case, the server also adds a corresponding entry to DMQ or PMQ, depending on which memory region the object is placed.

Object migration. On an object reference, a RaCC server updates the local ranking data structure that corresponds to the memory region where the object lies (DMQ for DRAM or PMQ for PCM). If the object reaches queue *PopThres* in PMQ, the server considers the object popular and tries to migrate it into its own DRAM. The migration occurs if the server has enough free DRAM space or unpopular objects in DRAM to replace. Lines 11 and 41 of Algorithm 3 are the two cases in which RaCC attempts to migrate a popular object from its PCM into its DRAM (*tryMigrate*). RaCC may reach line 11 when the initial node serves the request, and may reach line 41 when a remote node serves the request from its PCM. The migration occurs at the server that is caching the object. If the migration takes place, the server updates its PMQ and DMQ structures to reflect the migration and then broadcasts the fact that the object is now in DRAM. This broadcast is the only new communication introduced into PRESS.

Object replication and popularity hinting. The basic object replication mechanism of PRESS is not concerned about which type of memory will host a replica. RaCC introduces a hinting mechanism that helps concentrate popular objects in DRAM, avoids unnecessary technology migrations upon replication, and requires low-overhead maintenance. The basic idea is to allow RaCC to determine whether it should place a new replica in DRAM or PCM based on the object's popularity. Note that a server that is requested to host an object replica does not have any local information about the object's popularity in the other servers. To approximately infer the popularity of an object upon replication, RaCC looks up the GD to find if an object is in the DRAM of any server. If so, it caches the object in its own DRAM. Otherwise, it follows Algorithm 2 to place the replica. Over time, RaCC tends to concentrate popular objects in DRAM.

Modifications to the GD. In addition to the server-level policy modifications described before, RaCC requires GD extensions. The original GD maintains a hash table that maps each object ID to an entry representing the object. The object entry contains a pointer to a list of *object-server entries*, which represent the servers caching the object locally. When a server starts caching an object or replica, it creates a new object-server entry in the GD's object map. An object-server entry contains a pointer to an array (indexed by server ID) that maintains information about the server's load.

We extend each object-server entry with an *In-DRAM* single-bit flag, which indicates whether an object is cached in DRAM at that server. Additionally, when RaCC migrates an object into the DRAM of a server, the server broadcasts that fact, and all servers set the corresponding bit in their copy of the GD.

4.3.3 RaCC for Memcached

We study a simple server organization of Memcached, where Web servers (Memcached clients) receive requests and distribute them to Memcached servers. To distribute requests, the Web-tier servers hash the URLs of the requested objects and apply a MOD operation to find a target server in the Memcached tier. Due to this simple request distribution and the absence of replicas in Memcached, the opportunities for RaCC are limited. Specifically, for this service organization, we focus only on migrating popular objects within individual nodes. Interestingly, we observe that RaCC still improves service latency and robustness. Our results (Section 4.4) focus on the Memcached tier, where our approach applies.

```
function coopCacheServe (Request r, Peer peer)
begin
    Object obj = r.target
    if obj.size > MAX_SIZE then
        replyToPeer (peer, serveFromDisk (r))
    else
        if obj is not cached then
            startCaching (obj)
            replyToPeer (peer, serveFromCache (obj))
        else
            replyToPeer (peer, serveFromCache (obj))
            tryMigrate(obj)
        end
    end
end
```

Algorithm 4: RaCC request service for Memcached

Request service. Algorithm 4 depicts the Memcached behavior upon receiving a request, when augmented with RaCC. Like in PRESS, we only cache objects smaller than 1MByte (lines 3–5). The caching algorithm is straightforward. On a cache miss, a Memcached server brings the object to its local memory and creates the corresponding data structure in its DMQ or PMQ, depending on which memory region caches the object (lines 7–8). The server then replies to the same Web server that forwarded the request originally (lines 9, 11). The Web server forwards the reply to the service's client. The Memcached server updates the MQ structure corresponding to the memory region where the object is cached (line 10), then tries to migrate the object (line 12), as described below.

Object migration. Similarly to the local migration described for PRESS, when an object in PCM becomes popular at a Memcached server, the server considers migrating it into DRAM. The migration occurs if the server has enough free space or unpopular DRAM objects to accommodate the popular object. Unlike in PRESS, the server does not need to broadcast any messages.

4.4 Evaluation

In this section, we study PRESS and Memcached on clusters where server memories comprise only DRAM, only PCM, or a hybrid system with and without RaCC. We also evaluate each cluster with either SSDs or HDDs as storage devices.

4.4.1 Methodology

Workloads. We use real and synthetic HTTP traces from different sources. The *wiki* trace is a sample of actual Wikipedia traces [81]. The sample contains 2% of all user requests issued to Wikipedia (in all languages) in November of 2007. The *ircache* [41] trace contains actual requests collected in January of 2007 from 2 medium/large Web cache proxies located in California. The *govcomm* trace is a combination of several actual traces from the 1990's (ClarkNet, EPA, NASA, and World Cup '98) which we merged to create a larger data set and higher offered load. The *twitter* and *flickr* traces were synthetically generated using scripts and object popularity distributions provided by researchers at the University of Michigan [52]. The traces follow empirically measured and fitted distributions, respectively, from Twitter.com and Flickr.com. The popularity of a Twitter object (tweet) is inferred from their number of followers, and the popularity of Flickr pictures is proportional to their number of views.

Table 4.1 summarizes the main characteristics of our workloads. For each workload, the table shows: (1) the average and peak offered loads (in requests per second), (2) the average size of requests in KB, (3) the data sizes in GBytes and in number of objects, and (4) the average hit ratio when the system comprises solely either DRAM or PCM. Workload locality and cache sizes determine the hit ratio of the cooperative cache. Figure 4.1 depicts the cumulative distribution of requests to objects. The X-axis contains fractions of the total number of objects sorted from most to least popular. The Y-axis shows the corresponding percentage of requests directed to each fraction of the objects. For example, in wiki, a significant fraction of the requests concentrate on a relatively small number of objects, whereas in ircache, the distribution of requests increases almost linearly from 10% of the objects on. Because the workloads have different working set sizes (which are small in some cases) we use different memory sizes for each workload. Our baseline DRAM DIMM size is 32MBytes for govcomm and ircache, 192MBytes for twitter and flickr, and 1GBytes for wiki. Assuming 2 DIMMs per server and a 4-times-denser PCM, we obtain different cache sizes for each memory organization we study. In our simulations, a server for govcomm and ircache uses 64MBytes (DRAM-only), 160MBytes (hybrid memory systems), or 256MBytes (PCM-only); a server for twitter and flickr uses 384MBytes (DRAM-only), 960MBytes (hybrid memory systems), or 1536 MBytes (PCM-only); and a server for wiki uses 2GBytes (DRAM-only), 5GBytes (hybrid memory systems), or 8GBytes (PCM-only).

Simulation infrastructure. Because PCM hardware and DRAM + PCM hybrid systems are not yet widely available, we use simulation in our evaluation. We simulate a server cluster that receives requests directly from clients (e.g., via round-robin DNS) and may communicate with each other to serve those requests. The servers are connected to the clients via an external (public) network, and to each other via an internal (private)

Table 4.1: Workload details

Name	Reqs/s (avg / peak)	Avg. KB/req	Dataset (GB / objects)	DRAM- / PCM-only hit ratio
govcomm	876 / 3343	21.08	6.35 / 111907	99.2% / 99.9%
wiki	572 / 885	40.50	202.32 / 4856760	77.6% / 85.5%
twitter	1999 / 2344	1.03	7.46 / 977964	88.8% / 99.7%
flickr	1999 / 2344	24.93	3.55 / 128728	96.2% / 100%
ircache	20 / 233	77.94	51.29 / 644862	44.5% / 52.6%

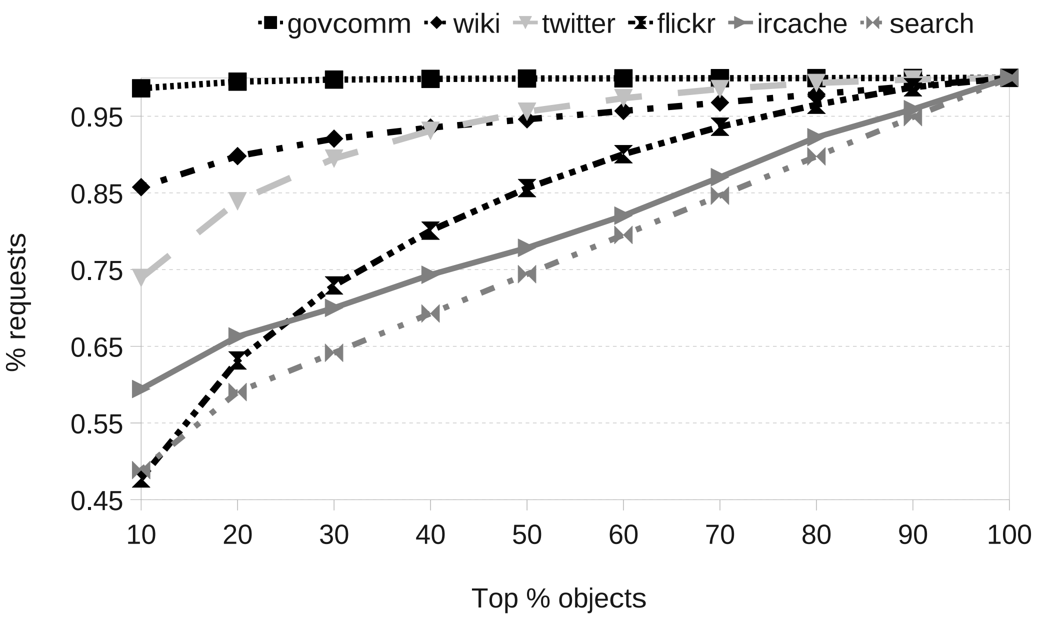


Figure 4.1: CDF of object popularity in our workloads.

network.

Our simulator reads the HTTP requests from a trace and replays them in open-loop. The simulator is event-driven and models each node of the cluster as a collection of inter-connected components. We characterize each component (i.e., CPU, SSD/HDD, NICs, and memory) by a service rate, a counter of idle processing elements, and a wait queue. Processing requests takes visiting different sequences of components depending on the actions necessary to serve the request. For example, a cache replacement causes a visit to the persistent storage component, which does not occur on a cache hit. The service rate specifies how fast a component can process units of work assigned to it. If all processing elements of a component are busy, it will store new incoming work units in its queue until a processing element becomes available. A state machine drives the progress of each request, retiring it after all necessary components have been visited. We keep track of object location in memory and count writes to PCM frames for endurance calculations. Finally, we evaluate the load of a server by looking at its amount of pending requests. For example, in PRESS, we determine overload by comparing that number to the overload threshold. We empirically chose a threshold of 250 requests, a single value that improves load balancing in most cases without being excessively conservative.

Our simulated environment is an 8-node server cluster. Each server comprises (1) one 8-core CPU running at 2.13GHz; (2) an SSD or a 15Krpm HDD; (3) a 1Gb/sec Ethernet network interface card (NIC) with a full-duplex link for communication with the public network; (4) a 4X EDR Infiniband NIC (with support for remote DMA) for the private network; (5) two single-rank memory DIMMs (DDR3-1866), each one lying on a its own 64-bit DDR3 933MHz channel. Each memory rank has 8 chips (x8 width), each of which has 8 banks. Table 4.2 summarizes the power and timing characteristics of our servers.

We validated our performance simulation for DRAM-only servers by running a Memcached server (*libmemcached5*) and a multi-threaded client (*memslap* from *libmemcached-tools*) on a single HP Proliant DL320G6 E5620 server (4-core 2.4GHz CPU, 6 GB of channel-interleaved DDR3 DRAM 1066MHz). We measured the average inter-arrival rate and the average response time per request for 8 different request arrival rates. We repeated each measurement 10 times on a quiet server, and verified that the CPU utilization remained under 60% (to avoid a CPU-overload bias) and that no request timed out. We used the real arrival rates to generate input traces for the simulator. We found the real response times to be within 14% of our simulations for 6% RBHR on average, as shown in Figure 4.2, suggesting that our simulations are accurate for the DRAM-only system. Unfortunately, we cannot perform a similar set of validation experiments for the other systems, as they utilize yet-to-be-commercialized PCM chips.

Table 4.2: Cluster node features

Component	Values and units
CPU request forward / migration (empirically)	500ns / 1μs
CPU idle / max power [35]	30W / 105W
SSD 4KB read [45]	25μs
SSD Idle / max power [45]	0.75W / 6W
HDD 4KB read [75]	2.4ms
HDD idle / max power [75]	4W / 7W
Infiniband latency [54] / 1KB transfer [34]	100ns / 80ns
Infiniband idle / max power [53]	11W / 13W
DDR3-1866 [47, 55, 68]	Values and units
Row buffer read / write latency	22.47 / 36.38 ns/64B
Row buffer read / write energy	0.93 / 1.02 pJ/bit
Active standby / precharge standby power	0.72 / 0.58 W/rank
DRAM array read / array write latency	13.91 / 13.91 ns
DRAM array read / array write energy	1.17 / 0.39 pJ/bit
PCM array read / array write latency	110 / 300 ns
PCM array read / array write energy	4.94 / 33.64 pJ/bit

Memory model. Our DDR3 and DRAM power and timing parameters are from [55, 83], and PCM parameters from [47, 68]. In our simulations, a memory rank can be in (1) the Active Standby state, when at least one of its banks is serving requests; or (2) the Precharge Power Down state, when all banks are idle and the clock enable line is turned off to save energy. Additionally, PCM is enhanced to avoid writing unmodified cache lines back to the cell array. The MC implements bank-level interleaving and serves memory requests in first-come first-served (FCFS) order. In the absence of row conflicts, the MC keeps banks open during a DMA operation. The CPU schedules DMA operations in FCFS order.

To be able to easily study a range of object-to-memory mappings, we parametrize the simulator with the expected row buffer hit ratio (RBHR). A low RBHR brings out the performance of the memory devices in each access, whereas a high RBHR brings the performance of DRAM and PCM closer to each other. Table 4.3 shows the performance and energy consumption of DRAM and PCM for different average RBHRs. Because bank interleaving reduces the row buffer locality (in exchange for better throughput) and the physical frames of the requested objects are unlikely to

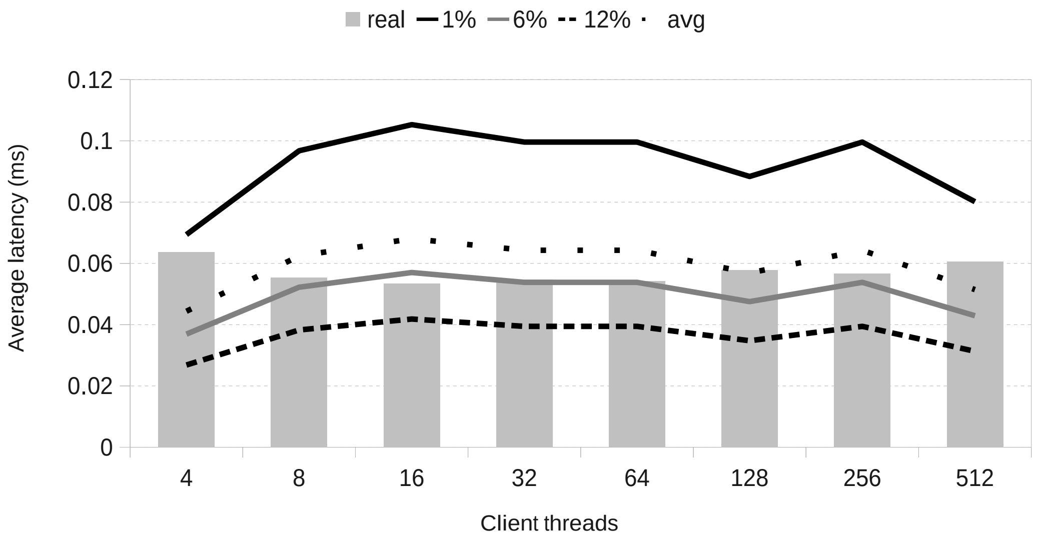


Figure 4.2: Response times for a real server, simulated servers (assuming 1%, 6%, and 12% RBHRs), and the average across the simulated response times.

be contiguous, we conservatively assume a low average row buffer hit ratio (12.5%) by default. For example, for every 8-KByte frame accessed in main memory, by default we read only 1KByte from each row buffer due to bank interleaving. For a commonly used 8-KByte row buffer and 8 banks, it is necessary to read $8 \times 8KB$ from the memory devices to be able to read or write our 8-KByte memory frames ($8KB \div 64KB = 12.5\%$ RBHR). Finally, multiple DMA streams may compete for memory (i.e., disk-to-memory, network-to-memory, memory-to-network), thus it is reasonable to expect even lower RBHR. In Section 4.4.2, we show a sensitivity study covering the expected range of average RBHR ratios.

Baselines for comparison. Because RaCC has been designed for server clusters with hybrid memory, we compare it to an "*Unmanaged*" hybrid memory cluster that does not explicitly differentiate between DRAM and PCM. We also compare the hybrid-memory systems to two baselines: DRAM-only and PCM-only clusters. These two baselines implement the original versions of PRESS and Memcached. We study the systems under different RBHRs.

We also consider the impact of the persistent storage devices on the service

Table 4.3: Influence of average RBHR on memory accesses [47, 68]

ns/8KB	1% RBHR	6% RBHR	12% RBHR
DRAM_R	3219	1216	882
DRAM_W	4109	1439	993
PCM_R	8479	2531	1540
PCM_W	28569	7553	4051
nJ/8KB	1% RBHR	6% RBHR	12% RBHR
DRAM_R	35447	4421	1867
DRAM_W	43427	4926	1997
PCM_R	96753	10911	4376
PCM_W	278970	24371	9399

performance. Recent studies have shown that complete replacement of HDDs with SSDs in datacenters is not cost-effective [44,60]. However, the same studies suggest that SSDs can be exploited as an intermediate tier between HDDs and DRAM, depending on the workloads' characteristics. In such an organization, we should expect performance to lie somewhere in between the SSD and HDD performance results we present.

4.4.2 Performance

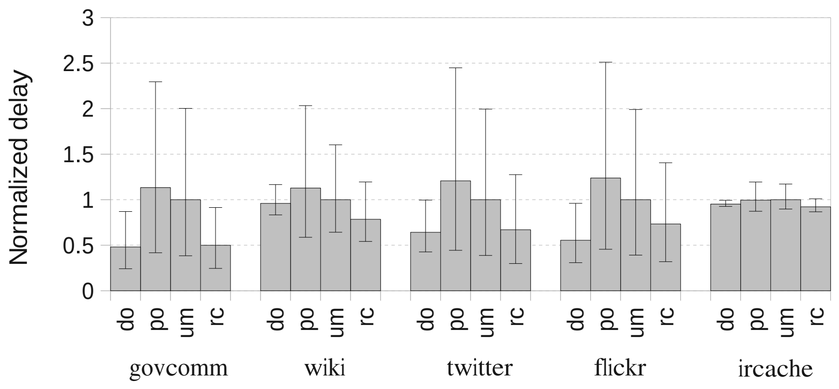
Performance of PRESS. Figure 4.3(a) shows the latency per request served from a PRESS cooperative cache, where each server uses a fast SSD as the persistent storage device. The bars in the figure represent the average latency per request across different RBHRs (1%, 6% and 12%). The upper and lower ends of the error bars represent the highest and the lowest latencies across RBHRs, respectively. In general, as we expected, the relative performance advantage of DRAM over PCM is higher in the presence of low RBHRs, because they expose more of the raw performance of the memory devices to the service. Analogously, high RBHRs hide the performance disadvantage of PCM. In fact, all maximum latencies in the figure consistently correspond to the lowest RBHRs, whereas the minimum latencies correspond to the highest RBHRs.

The SSD-based cluster is ideal for DRAM-only because (1) DRAM serves the cooperative cache hits with better performance than PCM; (2) cooperative cache misses have relatively low penalty, due to the high performance of the SSDs; and (3) most of our workloads have relatively high cooperative cache hit ratios (Table 4.1), which reduces the frequency of accesses to the storage devices. In fact, DRAM-only performs well with SSDs when its hit ratio is within 5% of the other techniques (govcomm, flickr, and ircache). The DRAM-only advantage diminishes beyond that hit ratio, or when the SSDs are slower than those we consider in this study. Regardless of the hit ratio, we note that RaCC comes within 1% on average of DRAM's ideal results, while performing better than the other systems.

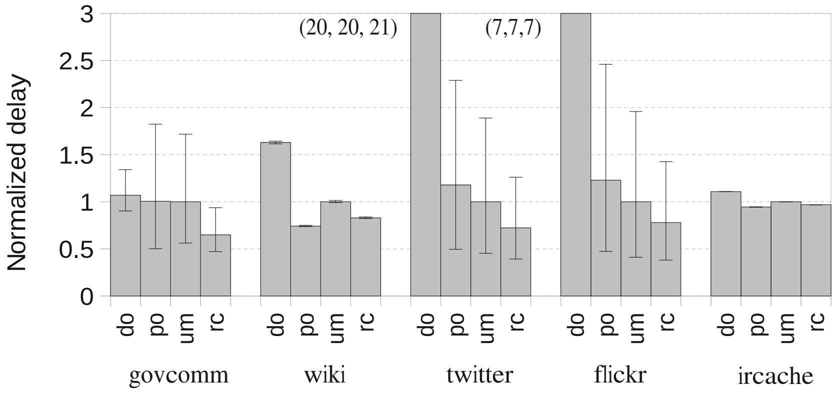
Although PCM-only exhibits the highest cooperative cache hit ratios across all workloads due to its large storage capacity, it performs worse than the other systems. The reasons are (1) the low miss penalty of the cooperative cache; (2) the inferior performance of PCM compared to DRAM (present in the other systems); and (3) the fact that the hybrid systems serve a significant fraction of their accesses from DRAM. For those reasons, compared to PCM-only, RaCC improves performance by 37% on average, for our workloads.

Finally, in this setup, RaCC performs better than Unmanaged by 28%, on average, for the workloads we study. The reason is that RaCC concentrates popular objects in DRAM, creating more opportunities to serve them from that memory region. We show that result in Figure 4.3(c), which quantifies the percentage of requests that the cluster serves from its SSDs (on cache misses), DRAM, and PCM (on cache hits). The figure illustrates that the RaCC serves substantially more requests from DRAM than Unmanaged, especially in the case of govcomm. The number of objects that RaCC migrates is below 1% of the total number of objects requested for all of our workloads.

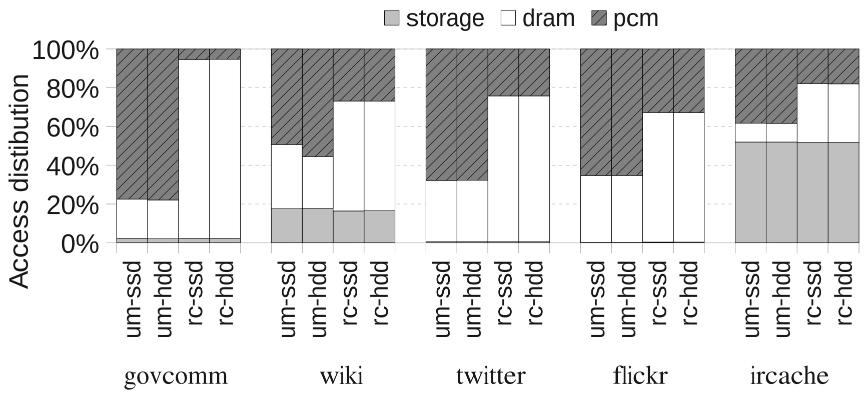
Figure 4.3(b) shows the latency per request served from PRESS with HDD-based servers. In this setup, RaCC performs better than DRAM-only, PCM-only, and Unmanaged, respectively by 87%, 23%, and 21% on average, for the workloads we study. Because random HDD reads are two orders of magnitude slower than SSD reads, the severe penalty of cooperative cache misses makes the hybrid and PCM-only systems much more attractive than DRAM-only. In fact, DRAM-only presents significantly lower performance compared to the others for twitter and flickr, despite being within 5% of the cache hit ratio of the other systems. RaCC performs better than PCM-only



(a) Request latency (PRESS+SSD)

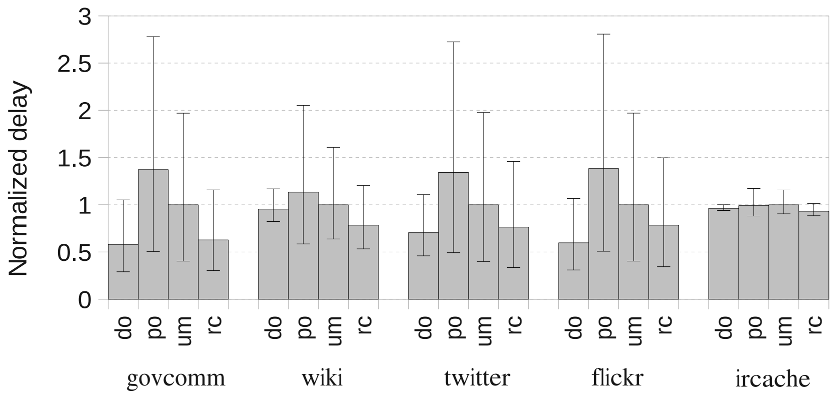


(b) Request latency (PRESS+HDD)

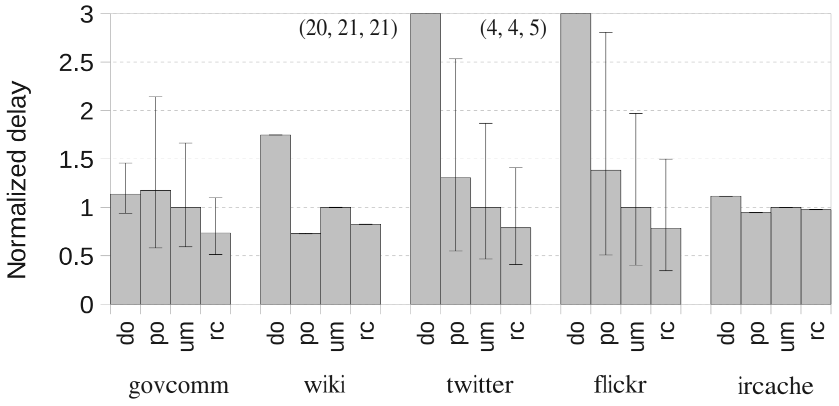


(c) Request service in PRESS

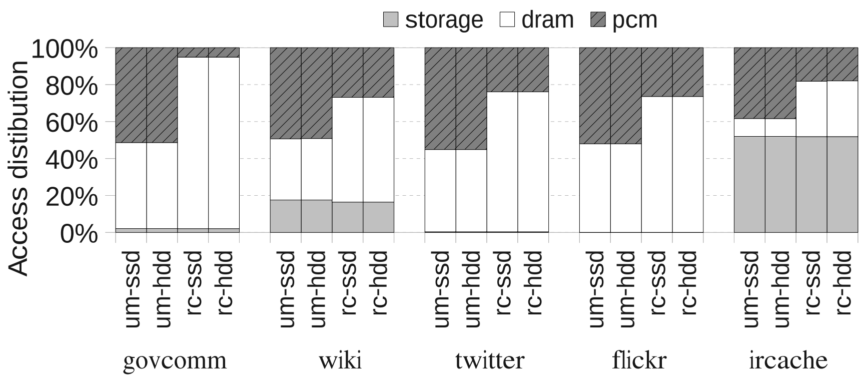
Figure 4.3: Average latency per request served by PRESS, using either SSD or HDD storage devices, and percentage of requests served from each memory and storage device. Each bar represents the average latency per served request across the multiple RBHR scenarios (1%, 6% and 12%) normalized to Unmanaged. The error bars show the worst and the best average latency per request across all RBHRs. Do, po, um, and rc represent respectively DRAM-only, PCM-only, Unmanaged, and RaCC.



(a) Request latency (Memcached+SSD)



(b) Request latency (Memcached+HDD)



(c) Request service in Memcached

Figure 4.4: Average latency per request served by Memcached, using either SSD or HDD storage devices, and percentage of requests served from each memory and storage device. Each bar represents the average latency per served request across the multiple RBHR scenarios (1%, 6% and 12%) normalized to Unmanaged. The error bars show the worst and the best average latency per request across all RBHRs. Do, po, um, and rc represent respectively DRAM-only, PCM-only, Unmanaged, and RaCC.

in all variations, except in wiki and ircache. However, the gain of PCM-only is small (10% and 2% respectively) in those cases.

As shown in Figure 4.3(c), RaCC is able to serve more requests from DRAM than Unmanaged, thus presenting better average performance. In this setup, the number of objects that RaCC migrates is also less than 1% of the objects requested in each workload, thereby not impacting the service latency noticeably. Additionally, we observe that, without the hinting feature in PRESS (Section 4.3.2), RaCC would perform poorly. The reason is that sets of popular objects are often responsible for a high load and, as a result, they tend to be replicated across many servers. Without hints, the replicated objects often replace cached content in PCM, then become popular and require migration into DRAM.

Performance in Memcached. Figures 4.4(a) and 4.4(b) show the performance per request served from an SSD- and an HDD-based Memcached cluster, respectively. At first glance, the Memcached clusters exhibits similar average performance as the PRESS clusters. However, a closer look reveals that Memcached exhibits higher standard deviation than PRESS in both request distribution and average latency per request across different servers. Specifically, Memcached's standard deviation in request distribution is 70% and 80% higher than PRESS using SSDs and HDDs, respectively. In terms of average latency per request, Memcached's standard deviation is 60% and 82% higher than PRESS using SSDs and HDDs, respectively. The limited load balancing of Memcached (only via hashing) makes server performance less deterministic and more prone to performance bottlenecks. However, the lack of replicas entails extra space in the individual memories for caching more objects. We observe that the cooperative cache hit ratio does not change significantly between PRESS and Memcached, despite their functional disparities. For that reason, load balancing is the main factor behind the absolute performance difference of DRAM-only and PCM-only in PRESS and Memcached. Compared to the other memory organizations, Unmanaged Memcached performs better than its PRESS counterpart. The reason is that Unmanaged PRESS often replicates objects in PCM, and because most replicated objects are popular, PCM ends up serving more requests, as we observe in Figure 4.4(c). In contrast, RaCC is able to concentrate most accesses in DRAM by migrating popular objects to that memory region.

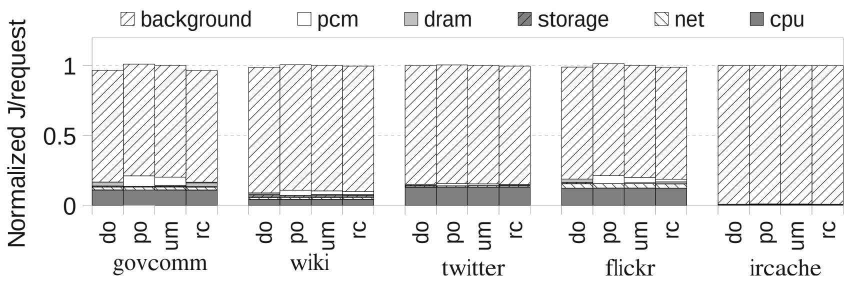
In the SSD-based clusters, across all workloads, RaCC's average latency lies within 2% of that of DRAM-only. RaCC also improves the average latency per request by 37% and 22%, respectively, compared to PCM-only and Unmanaged. In the HDD-based clusters, the performance advantage of RaCC over its competitors is 86%, 26%, and 18%, respectively, for the workloads we study.

In summary, we observe that the performance of the studied cooperative caches is highly influenced by the storage device, memory performance, RBHR, and the cooperative cache hit ratio. Across all these variants, RaCC presents the most robust performance. The total number of migrations is small compared to the total number of objects requested, staying below 1% in all cases.

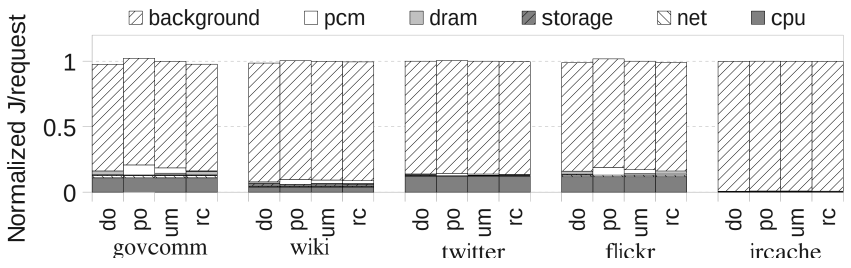
4.4.3 Energy

In this section, we compare the energy consumption of the studied memory organizations for both PRESS and Memcached. Specifically, we consider the static (background) and dynamic (non-background) energies of the memories and storage devices, plus the dynamic energy of CPU and NICs in the entire cluster. We omit the static energy consumption of CPU, NICs, power supplies, and network switches because although non-negligible, they are irrelevant to our study. For the same reason, we do not include the dynamic CPU and NIC energy involved in receiving and parsing client requests. The energy results we report assume 1% RBHRs because that scenario brings out the memory energy more than the others. Because migrations involve both cores and memory, the migration energy is included in the CPU and memory components. However, the migration energy is negligible in all cases.

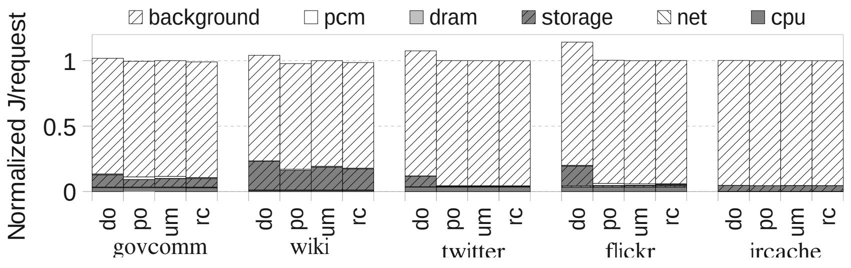
Figures 4.5(a) and 4.5(b) show the breakdown of energy consumption for SSD-based clusters managed with PRESS and Memcached, respectively. We observe that the static energy of the servers in periods of low and high utilization adds up and dominates the total energy consumption. Overall, the total energy per request is similar across memory organizations, all within 2% of each other. In fact, the dynamic energy of



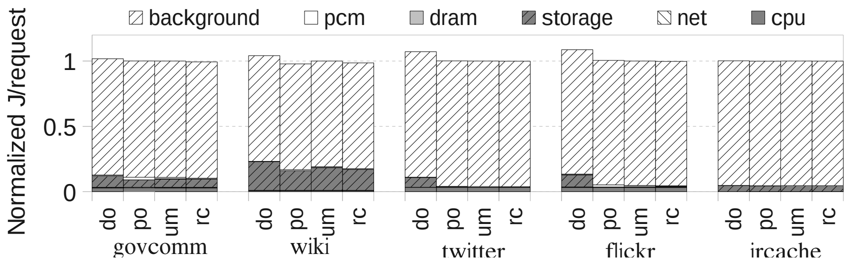
(a) Energy/request (PRESS+SSD)



(b) Energy/request (Memcached+SSD)



(c) Energy/request (PRESS+HDD)



(d) Energy/request (Memcached+HDD)

Figure 4.5: Energy consumption per request served by PRESS and Memcached, using either SSD or HDD storage devices. We calculate energy per request as total enegy divided by the number of requests served. Do, po, um, and rc represent respectively DRAM-only, PCM-only, Unmanaged, and RaCC.

RaCC is always close to that of DRAM-only (within 3%), which exhibits the best energy consumption per request served. The reason is the low energy penalty of cooperative cache misses and the lower energy consumption of DRAM compared to PCM. Although PCM consumes lower idle power than DRAM, its higher access latency leads to higher energy consumption. Because RaCC concentrates most accesses in DRAM, it exhibits better dynamic energy than Unmanaged and PCM-only by 9% and 13% in PRESS clusters, and by 8% and 14% in Memcached clusters on average. As we noted above, the number of migrations in RaCC is very small, and thus their energy overhead is low.

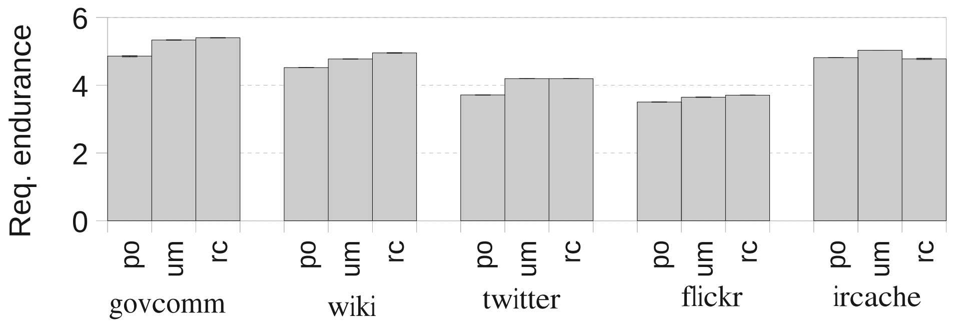
Figures 4.5(c) and 4.5(d) show the breakdown of the energy consumption for HDD-based clusters managed with PRESS and Memcached, respectively. Like in the SSD-based clusters, we notice that the static energy is much more significant than the dynamic energy. However, in this scenario, the energy penalty of cooperative cache misses is high and the static and dynamic energy of the storage devices dominates the total energy per request served. In fact, RaCC exhibits total energy consumption within 1% of Unmanaged and PCM-only. Because of the high penalty of cache misses, DRAM-only is slightly worse than RaCC by 6% for PRESS and 5% for Memcached. Comparing only the dynamic energy, on average, we find that RaCC is only slightly better than Unmanaged (within 4%) and PCM-only (within 5%), but significantly better than DRAM-only (49% for PRESS and 47% for Memcached).

4.4.4 Endurance

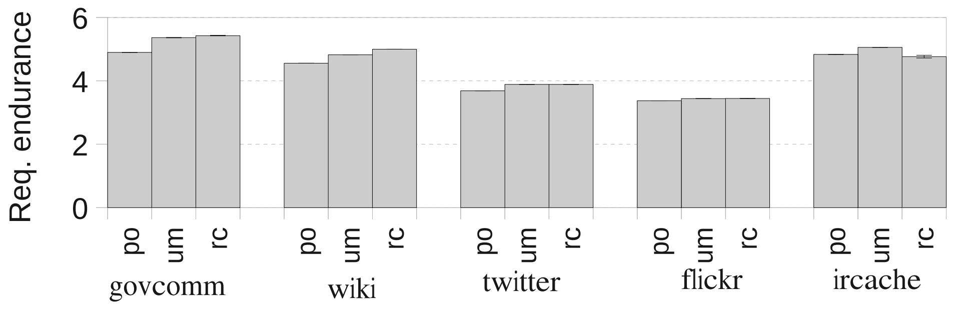
In this section, we compare the endurance of PCM in the systems we study. As in Section 3.3.2.2, we use the Required Endurance metric [11] over a period of 5 years.

Figures 4.6(a) and 4.6(b) compare the base-10 logarithm of the Required Endurance of the systems we consider, using the 5-year projection. For comparison, PCM's endurance today is $10^8 - 10^9$. As shown in the figures, all systems exhibit relatively low PCM endurance requirements, due to their limited memory write throughput and large PCM capacity. In both PRESS and Memcached, RaCC requires roughly the same endurance as Unmanaged (within 1% of each other). Because PCM-only has larger capacity on average, it requires 7% and 5% less PCM endurance than both hybrid systems, respectively in PRESS and Memcached. The larger capacity of PCM-only entails a higher value of C, but also improves the hit ratio of the cooperative cache, which increases the write traffic to the PCM of the hybrid systems.

As these results clearly show, PCM endurance is not a problem for the clusters and workloads we consider. In addition, like RaPP, RaCC can be combined with existing wear-leveling solutions for PCM to improve the endurance results presented in this section.



(a) Required endurance for PRESS



(b) Required endurance for Memcached

Figure 4.6: Required endurance of PCM for PRESS and Memcached over a period of 5 years in logarithmic scale. The solid bars represent the average across results for SSD and HDD storage devices. The error bars represent the maximum and the minimum values of required endurance. Do, po, um, and rc represent respectively DRAM-only, PCM-only, Unmanaged, and RaCC.

4.5 Summary

In this chapter, we introduced RaCC, an object placement policy for server clusters that use hybrid DRAM+PCM main memory systems. In particular, we integrated RaCC tightly into two existing cooperative caching systems. RaCC seeks to benefit from the best characteristics of DRAM and PCM, while avoiding their limitations. RaCC ranks, migrates, and replaces cached objects to improve the cache's performance. Our results for five workloads, four memory systems, and two types of persistent storage devices demonstrate that RaCC provides robust performance to hybrid-memory clusters, without increasing energy consumption. We conclude that the combination of hybrid memories and intelligent object placement can produce efficient and robust server clusters.

Chapter 5

Related Work

5.1 Using technologies other than DRAM for main memory

Despite the problems with Flash (page-level interface of NAND Flash, very high write and block-erase times, low cell endurance), two studies have considered combining Flash and DRAM in main memory. ENVy [84] focused on sidestepping the write-related problems of Flash using battery-backed SRAM and virtual memory techniques. A more recent position paper [59] considered a flat DRAM+Flash (or PCM) address space with the OS responsible for predicting page access patterns and migrating read-only pages from DRAM to (write-protected) Flash. Although the paper did not include an evaluation of their system, we expect that an OS-only approach to page management would cause unacceptable overhead for most types of pages.

With better characteristics than Flash, PCM is a more promising technology for use in main memory. In fact, several recent works [13, 16, 47, 67, 69, 70, 87, 89, 90] have proposed systems that use PCM as a partial or complete replacement for DRAM. In this dissertation, we compared RaPP to two of these works.

Qureshi *et al.* [67] used a hardware-managed DRAM buffer between the LLC and a PCM-only main memory. The buffer allows direct application of fine-grained wear-leveling, lazy writes, write coalescing, and fine-grain write-backs to PCM. Zhang *et al.* [89] placed DRAM and PCM in a flat address space and treated DRAM as an OS-managed write partition. In their system, all pages are initially stored in PCM. The idea is to keep the cold-modified (infrequently written) pages in PCM, trying to take advantage of its low idle power consumption, and the hot-modified (frequently written) pages in DRAM to avoid PCM's high write latency and poor endurance. The solution entails a hardware implementation of MQ for page ranking only. In contrast, RaPP may migrate pages during the OS quantum and disable itself if most migrations are useless. Wu *et al.* [85] proposed different implementations of hybrid cache hierarchies, combining different memory technologies (including DRAM and PCM) in CPU cache hierarchies. One of their policies considers access frequency for cache block placement. Bheda *et al.* [7] replace row buffers in PCM devices with small fully-associative caches made of eDRAM and managed by the MC. Their design consumes lower power and performs better than large DRAM and PCM arrays. Qureshi *et al.* [68] adopted a PCM-only memory comprising 2 PCM areas with different densities. Their system dynamically reconfigures PCM to resize those areas on demand and attain higher performance (low-density PCM) or larger storage (high-density PCM).

Although RaCC also manages data placement in a flat hybrid memory organization, it differs from previous approaches for at least three reasons. First, RaCC's management spans multiple servers. Second, RaCC manages objects that span one or multiple frames, which potentially reduces the bookkeeping overheads compared to hardware-based approaches. Third, RaCC is completely implemented in software, with few OS modifications, for hybrid-memory servers.

5.2 Tackling PCM's Endurance Problem

Many previous works focused extensively on the workarounds needed to mitigate this problem. Lee *et al.* [47] proposed tracking data modifications at the cache block and data word levels to avoid unnecessary traffic to the MC and writes to the PCM array. The technique was combined with narrow row buffer sets organized to exploit locality.

Yang *et al.* [87] proposed the Data-Comparison Write (DCW) approach, which only allows a write to a PCM cell if the new value differs from the previously stored one. Flip-and-Write [16] improves DCW by storing extra "flip bits" that denote the encoding of each PCM word and performing hamming distance calculations to verify if reversing the encoding (by inverting flip bits) will reduce the number of writes. Alternatively, Zhou *et al.* [90] improved DCW using periodic byte rotation at the cache block level and segment swaps across memory areas.

A complementary technique named Fine-Grained Wear-Leveling (FGWL) seeks to balance the wear-out across cache blocks within a physical PCM frame by rotating blocks within the frame [67]. FGWL inspired Start-Gap [69], a technique that applies a simple algebraic function to transform a logical memory address into a physical one. [69] also combined Start-Gap with simple address-space randomization techniques.

Another approach [40] improves endurance by reusing a pair of physical frames with complementary faulty cells as a single logical frame. The system creates the pairs periodically, as new faults may alter previous pairings. A related approach detects and corrects faulty PCM cells by encoding the location of the faulty cell into a pointer and allocating extra cells to store the correct content [74]. Writes to PCM occur once per fault, improving endurance over error correction codes.

Differently than these previous systems, RaPP takes a higher level, page-based approach to wear leveling. First, we migrate write-intensive pages to DRAM. Second, we migrate pages coming from DRAM to unpopular PCM frames. Importantly, their low-level techniques are orthogonal and can be nicely combined with our page-based techniques to extend endurance further.

RaCC is also orthogonal to many of these wear-leveling solutions and can be combined with one or more of them to improve PCM's lifetime, which justifies our shift towards improving the performance of PCM.

5.3 Page Migration in Main Memory

A few works have considered page migration for memory energy conservation. Lebeck *et al.* [46] conducted a preliminary investigation of popularity-based page allocations to enable sending (unpopular) memory devices to low-power state. In [32], the OS migrates pages periodically based on their reference bits, without any support from the MC. In contrast, Pandey *et al.* [65] proposed to implement popularity-based page ranking and migration using the MC to reduce the energy consumed by DMA transfers. Dong *et al.* [23] migrate hot pages from the off-chip memory to a chip-integrated memory. In a position paper [5], Bellosa proposed "memory management controllers" (MMCs) that would take away the responsibility for memory management (e.g., page migration) from the OS.

An OS-only approach to page migration can improve performance for I/O-intensive workloads, as we observe in RaCC, and even work for energy conservation, as in [32]. However, for memory-intensive workloads, an OS-based approach does not react quickly and efficiently enough to mitigate the problems with PCM. Thus, in this case, it is critical to involve the MC (or an MMC) as well. In this context, the closest prior work to RaPP is [65]. However, as our environment (a DRAM+PCM hybrid in a multiprocessing system) and goal (improve performance at the same energy consumption) are quite different than theirs (DRAM-only memory and DMA-related energy conservation in a uniprocessor system), there are many differences between the two approaches. First, we had to adapt a sophisticated ranking algorithm to address the poor performance and endurance of PCM. Pandey *et al.* used a much simpler histogram-based ranking. Second, our migration approach is also more complex, as it needs to consider the limitations of PCM and involve an additional, properly selected PCM frame. Third, we considered many multiprocessing workloads and the increased pressure they put on the memory system. Because of this pressure, migrations need to be done very selectively, so we had to devise heuristics to avoid certain migrations.

5.4 Cooperative Caching

Several locality-aware cooperative caches have been proposed in the literature [2, 3, 8, 12, 15, 26, 64]. However, this dissertation is the first to consider PCM in the context of server clusters. In fact, our approach is directly applicable to all of these systems, although they may entail specific implementation challenges.

Pai *et al.* proposed the first locality-aware server cluster [64]. The cluster features a central front-end load balancer that receives and hands off client requests (based on their content) to back-end servers. Carrera and Bianchini made the hand-off mechanism distributed by leveraging user-level communication and fast networks [8, 12]. Chiu *et* *al.* proposed a cooperative cache for cloud environments where large caches entail high costs for the hosted service [15]. Barely-alive cooperative caches [2] reduce idle energy consumption by turning off most components of idle servers, while keeping their memories active and accessible via a special NIC CPU and a distributed middleware. Another approach [3] manages servers built with low-power components to improve performance per Joule. Memcached is currently used to implement cooperative caches in mainstream Internet services, such as Facebook, Youtube, and Wikipedia [26].

Chapter 6

Conclusion

In this dissertation, we sought to improve the performance of server hybrid memory systems that combine DRAM and PCM, without increasing the memory energy consumption significantly. Hybrid systems typically exploit the high performance of DRAM, and the high scalability and large capacity of PCM, while mitigating the disadvantages of both technologies. We proposed and evaluated two novel hybrid memory systems that concentrate popular data in DRAM, while still allowing direct access to data stored in PCM. Our first hybrid system is meant for individual servers running memory-intensive workloads, whereas our second system is meant for server clusters running I/O-intensive workloads.

For individual servers, we proposed a hardware-driven design that features a sophisticated memory controller, and a page ranking and migration policy called RaPP. The memory controller monitors the memory accesses and implements RaPP. The policy takes into account the recency and frequency of page accesses, as well as the write traffic to each page, to rank pages and lay them out in physical memory. Our results demonstrate that our design behaves significantly better than two state-of-the-art hybrid designs, despite our optimistic assumptions for the latter systems.

For server clusters that use hybrid DRAM+PCM main memory systems, we proposed a software-driven approach, called RaCC. In particular, we integrated RaCC tightly into two existing cooperative caching systems. RaCC ranks, migrates, and replaces cached objects to improve the cache's performance. Our results for five workloads, four memory systems, and two types of persistent storage devices demonstrate that RaCC provides robust performance to hybrid-memory clusters, without increasing energy consumption. This is the first study of the implications of PCM and hybrid main memory systems for server clusters.

We conclude that PCM is a promising main memory technology for meeting the fast increasing need for memory capacity. As the technology matures, its cost-per-bit, power consumption, and heat dissipation (important design constraints for modern systems and datacenters) should be lower than that of DRAM. For easier adoption in main memory, PCM can be combined with small amounts of DRAM. However, for best performance and robustness, those hybrid systems require careful data placement and migration.

We expect that the advent of hybrid memory systems will create many new avenues for research, including how the hybrid nature of the system can be exposed to programmers, how operating systems can best take advantage of the two types of memory, and memory controller designs that are tailored for PCM. In addition, PCM's non-volatility creates opportunities for new uses of main memory.

6.1 Future Research

In this section, we list some examples of interesting future work.

- Some of our assumptions about the memory controller could be different from those presented in Chapter 3. For example, the memory access scheduling, address mapping (from physical address into channel, bank, rank, row, and column), row buffer management, row buffer size, and data interleaving, can play a role in memory performance, energy, and endurance. For memory-intensive workloads, one could investigate the impact of these factors in hybrid systems.

- One could also study the impact of higher memory request arrival rates, which could be expected from the use of manycore CPUs, out-of-order execution, and multi-threaded workloads.

- In this dissertation, we focused on incorporating PCM into an existing memory interface (DDR3). However, one could also rethink the architecture of the memory interface to make better use of PCM. Along these lines, [7] proposed hybrid PCM chips, where a small eDRAM cache replaced the row buffers; [47] modified the CPU, the memory controller and the row buffers to leverage PCM; and [22] and [21] propose data prefetching structures at the PCM peripheral circuitry. Rearchitecting memory may also lead to better synergy between PCM and the high-throughput, low-power interfaces of the future, such as DDR4 [42].

- In the context of cooperative caches, one could, for example, (1) exploit the non-volatility of PCM to reduce idle energy (e.g., by turning off nodes or memory ranks, without losing the cached content); (2) study our object-placement policy in a reconfiguring Memcached cluster (when nodes leave and enter the cluster) or when cached objects can be segmented into (cacheable blocks) that are mapped to different servers.

- Another interesting research is to prototype hybrid memory systems, and implement our approaches in real environments. A step towards this goal has been taken in [1], where PCM-based DIMMs and a PCM-enabled memory controller have been demonstrated. However, those components were not used in main memory, but in an SSD. Prototyping those systems is made difficult by the trend of embedding memory controllers in the CPUs. In addition, as PCM chips mature, its operation frequency may still lag, which could waste resources or require adaptation when used with cutting-edge memory interfaces. However a study with hybrid systems using an external controller should still provide insights and evidence to motivate embedded controllers with support to hybrid systems.

- Finally, as new materials, such as graphene, are discovered and advance in development, new challenges and benefits are expected also in future memories. In fact, future ITRS projections already describe the future replacements of all memory technologies targeted in this dissertation, including PCM. Independently from the search for new materials and manufacturable solutions, memory research should continue to look for scalable, low-power, and high throughput solutions.

References

[1] Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson. Onyx: A Protoype Phase-Change Memory Storage Array. In *Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2011.

[2] Vlasia Anagnostopoulou, Susmit Biswas, Alan Savage, Ricardo Bianchini, Tao Yang, and Frederic T. Chong. Energy Conservation in Datacenters through Cluster Memory Management and Barely-Alive Memory Servers. *Workshop on Energy-Efficient Design (WEED)*, 2009.

[3] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Symposium on Operating Systems Principles (SOSP)*, 2009.

[4] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan and Claypool Publishers, 2009.

[5] F. Bellosa. When Physical Is Not Real Enough. In *ACM SIGOPS European Workshop*, 2004.

[6] A. Bhattacharjee and M. Martonosi. Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[7] R.A. Bheda, J.A. Poovey, J.G. Beu, and T.M. Conte. Energy efficient phase change memory based main memory for future high performance systems. In *International Green Computing Conference (IGCC)*, 2011.

[8] Ricardo Bianchini and Enrique V. Carrera. Analytical and Experimental Evaluation of Cluster-based Network Servers. In *World Wide Web (WWW)*, volume 3, 2000.

[9] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 26(4), 2006.

[10] Daniel Bowers. You Probably Don't Need More DIMMs, 2010. h30507.www3.hp.com/t5/Eye-on-Blades-Blog-Trends-in/You-Probably-Don-t-Need-More-DIMMs/ba-p/81647.

[11] Geoffrey W. Burr, Matthew J. Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bulent Kurdi, Chung Lam, Luis A. Lastras, Alvaro Padilla, Bipin Rajendran, Simone Raoux, and Rohit S. Shenoy. Phase change memory technology. *Journal of Vacuum Science & Technology B*, 2010.

[12] Enrique V. Carrera and Ricardo Bianchini. Press: A clustered server based on user-level communication. In *IEEE Transactions on Parallel Distributed Systems*, 2005.

[13] A.M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, Jiahua He, A. Jagatheesan, R.K. Gupta, A. Snavely, and S. Swanson. Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2010.

[14] Ethan Chen and Tones Yen. Comparing SLC and MLC Flash Technologies and Structure, 2009. www.advantech.com.tw/epc/newsletter/Whitepaper/WhitePaper_Comparing_SLC_and_MLC_Flash_Technologies_and_Structure_200909.pdf.

[15] David Chiu, Apeksha Shetty, and Gagan Agrawal. Elastic Cloud Caches for Accelerating Service-Oriented Computations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2010.

[16] S. Cho and H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[17] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. Better I/O Through Byte-Addressable, Persistent Memory. In *Symposium on Operating Systems Principles (SOSP)*, 2009.

[18] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS European Workshop*, 2007.

[19] Bruno Diniz, Dorgival Guedes, Wagner Meira, Jr., and Ricardo Bianchini. Limiting the Power Consumption of Main Memory. In *International Symposium on Computer Architecture (ISCA)*, 2007.

[20] In Hwan Doh, Young Jin Kim, Jung Soo Park, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Towards greener data centers with storage class memory: minimizing idle power waste through coarse-grain management in fine-grain scale. In *Conference On Computing Frontiers*, 2010.

[21] X. Dong, N. Jouppi, and Y. Xie. PCRAMsim: System-Level Performance, Energy, and Area Modeling for Phase-Change RAM. In *The International Conference on Computer-Aided Design (ICCAD)*, 2009.

[22] Xiangyu Dong, Naveen Muralimanohar, Norm Jouppi, Richard Kaufmann, and Yuan Xie. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2009.

[23] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. Simple but Effective Heterogeneous Main Memory with On-Chip Memory

Controller Support. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010.

[24] Kamran Eshraghian. Evolution of Nonvolatile Resistive Switching Memory Technologies: The Related Influence on Hetrogeneous Nanoarchitectures. In *Transactions on Electrical and Electronic Materials*, 2010.

[25] EverSpin Technologies. INDUSTRIAL AUTOMATION - Customer Example - Siemens, 2011. www.everspin.com/technology.php?qtype=10.

[26] Brad Fitzpatrick. Distributed caching with memcached. In *Linux Journal*, 2004.

[27] Brinda Ganesh, Aamer Jaleel, David Wang, and Bruce Jacob. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2007.

[28] John F. Gantz, David Reinsel, Christopher Chute, Wolfgang Schlichting, John McArthur, Stephen Minton, Irida Xheneti, Anna Toncheva, and Alex Manfrediz. The Expanding Digital Universe - A Forecast of Worldwide Information Growth Through 2010, 2007.

[29] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, and Xiaodong Zhang. Does Internet Media Traffic Really Follow Zipf-like Distribution? In *ACM Special Interest Group on Performance Evaluation (SIGMETRICS)*, 2007.

[30] X. Guo, E. Ipek, and T. Soyata. Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[31] Yenpo Ho, Garng M. Huang, and Peng Li. Nonvolatile Memristor Memory: Device Characteristics and Design Implications. In *The International Conference on Computer-Aided Design (ICCAD)*, 2009.

[32] Hai Huang, Padmanabhan Pillai, and Kang G. Shin. Design and Implementation of Power-Aware Virtual Memory. In *USENIX Annual Technical Conference*, 2003.

[33] Hai Huang, K.G. Shin, C. Lefurgy, and T. Keller. Improving Energy Efficiency by Making DRAM Less Randomly Accessed. In *The International Symposium on Low Power Electronics and Design (ISLPED)*, 2005.

[34] InfiniBand Trade Association. InfiniBand Roadmap, 2011. www.infinibandta.org/content/pages.php? pg=technology_overview.

[35] Intel. Intel Xeon Processor E7-8830 (24M Cache, 2.13 GHz, 6.40 GT/s Intel QPI), 2011. ark.intel.com/products/53677/.

[36] International Technology Roadmap for Semiconductors (ITRS). Emerging Research Devices. www.itrs.net/Links/2007ITRS/ 2007_Chapters/2007_ERD.pdf, 2007.

[37] International Technology Roadmap for Semiconductors (ITRS). Emerging Research Devices, 2009. www.itrs.net/links/2009itrs/2009chapters_2009tables/2009_PIDS.pdf.

[38] International Technology Roadmap for Semiconductors (ITRS). 2010 Update Overview, 2010. www.itrs.net/links/2010itrs/2010Update/ToPost/2010_Update_Overview.pdf.

[39] E. Ipek, O. Mutlu, J.F. Martinez, and R. Caruana. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *International Symposium on Computer Architecture (ISCA)*, 2008.

[40] Engin Ipek, Jeremy Condit, Edmund B. Nightingale, Doug Burger, and Thomas Moscibroda. Dynamically Replicated Memory: Building Reliable Systems From Nanoscale Resistive Memories. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[41] IRCACHE, the NLANR Web Caching Project. IRCache Home. www.ircache.net, 2010.

[42] JEDEC. Main memory: DDR3 & DDR4 SDRAM, 2011. www.jedec.org/category/technology-focus-area/main-memory-ddr3-ddr4-sdram.

[43] Eun-ki Kim, Hyungjong Shin, Byung-gil Jeon, Seokhee Han, Jaemin Jung, and Youjip Won. FRASH: hierarchical file system for FRAM and flash. In *International Conference on Computational Science and Its Applications (ICCSA)*, 2007.

[44] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. HybridStore: A Cost-Efficient, High- Performance Storage System Combining SSDs and HDDs. In *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2011.

[45] Andrew Ku. Meet Intel's SSD 320, The Postville Refresh. www.tomshardware.com/reviews/intel-ssd-320-crucial-m4-realssd-c400,2908-2.html, 2011.

[46] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power Aware Page Allocation. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[47] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Architecture. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[48] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller. Energy Management for Commercial Servers. *IEEE Computer*, 36(12), 2003.

[49] Dean L Lewis and Hsien-Hsin S Lee. Architectural evaluation of 3D stacked RRAM caches. *IEEE International Conference on 3D System Integration*, 2009.

[50] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[51] Jiang Lin, Hongzhong Zheng, Zhichun Zhu, Howard David, and Zhao Zhang. Thermal Modeling and Management of DRAM Memory Systems. In *International Symposium on Computer Architecture (ISCA)*, 2007.

[52] David Meisner and Thomas F. Wenisch. Thin servers with fat pipes: architecting cost-effective data center Memcache systems. In *Technical report, Michigan University*, 2011.

[53] Mellanox Technologies. Adapter card product guide, 2009. www.colfaxdirect.com/store/pc/catalog/DDR_Bundle1.pdf.

[54] Mellanox Technologies. Voltaire Grid Director 4036E, 2011. www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/ voltaire_grid_director_4036E.

[55] Micron. 1Gb: x4, x8, x16 DDR3 SDRAM Features. download.micron.com/pdf/datasheets/dram/ddr3/ 1Gb_DDR3_SDRAM.pdf, 2006.

[56] Micron. TN-29-19: NAND Flash 101 NAND vs. NOR Comparison, 2006.

[57] Micron. 128Mb Parallel PCM: NP8P128A13TSM60E, 2011. www.micron.com/products/ProductDetails.html?product=products/pcm/ parallel_pcm/NP8P128A13TSM60E.

[58] Asit K. Mishra, Xiangyu Dong, Guangyu Sun, Yuan Xie, Narayanan Vijaykrishnan, and Chita R. Das. Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs. In *International Symposium on Computer Architecture (ISCA)*, 2011.

[59] Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, and Paolo Faraboschi. Operating System Support for NVM+DRAM Hybrid Main Memory. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[60] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to SSDs: analysis of tradeoffs. In *The European Professional Society on Computer Systems (EuroSys)*, 2009.

[61] Ashim Neogy. First Phase-Change Memory Cellphone, 2011. techmento.com/2011/05/09/phase-change-memory-cellphone/.

[62] T. Nirschl, J.B. Phipp, T.D. Happ, G.W. Burr, B. Rajendran, M.-H. Lee, A. Schrott, M. Yang, M. Breitwisch, C.-F. Chen, E. Joseph, M. Lamorey, R. Cheek, S.-H. Chen, S. Zaidi, S. Raoux, Y.C. Chen, Y. Zhu, R. Bergmann, H.-L. Lung, and C. Lam. Write Strategies for 2 and 4-bit Multi-Level Phase-Change Memory. In *IEEE International Electron Devices Meeting (IEDM)*, 2007.

[63] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. The case for RAMClouds: scalable high-performance storage entirely in DRAM. In *ACM SIGOPS Operating Systems Review*, volume 43, 2010.

[64] Vivek Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998.

[65] Vivek Pandey, W. Jiang, Y. Zhou, and R. Bianchini. DMA-Aware Memory Energy Management. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2006.

[66] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. Using Simpoint for Accurate and Efficient Simulation. In *ACM Special Interest Group on Performance Evaluation (SIGMETRICS)*, 2003.

[67] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[68] Moinuddin K. Qureshi, Michele M. Franceschini, Luis A. Lastras-Montaño, and John P. Karidis. Morphable memory system: a robust architecture for exploiting multi-level phase change memories. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[69] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[70] Luiz Ramos, Eugene Gorbatov, and Ricardo Bianchini. Page Placement in Hybrid Memory Systems. In *International Conference on Supercomputing (ICS)*, 2011.

[71] S. Rixner, W.J. Dally, U.J. Kapasi, P. Mattson, and J.D. Owens. Memory Access Scheduling. In *International Symposium on Computer Architecture (ISCA)*, 2000.

[72] Arthur Sainio. Strategy can maximize profit in DRAM, 2011. www.eetimes.com/design/memory-design/4215214/Strategy-can-maximize-profit-in-DRAM.

[73] Samsung Electronics. Samsung Introduces the Next Generation of Nonvolatile Memory - PRAM, 2006. www.physorg.com/pdf77297204.pdf.

[74] Stuart Schechter, Gabriel H. Loh, Karin Straus, and Doug Burger. Use ecp, not ecc, for hard failures in resistive memories. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[75] Seagate. Savvio 15K.2 - Highest-performing, greenest drive for enterprise storage systems - Data sheet, 2010. www.seagate.com/docs/pdf/datasheet/disc/ds_savvio_15k_2.pdf.

[76] Lei Shi, Zhimin Gu, Lin Wei, and Yun Shi. An Applicative Study of Zipf's Law on Web Cache. *International Journal of Information Technology*, 12(4), 2006.

[77] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[78] F. Tabrizi. The Future of Scalable STT-RAM as a Universal Embedded Memory, 2007. www.eetimes.com/design/embedded/4026000/The-future-of-scalable-STT-RAM-as-a-universal-embedded-memory.

[79] Texas Instruments. FRAM Applications, 2011. www.ti.com/ww/en/mcu/fram_ultra_low_power_embedded_memory/fram_mcu_applications.htm.

[80] Toshiba. NAND vs. NOR Flash memory, 2006.

[81] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia Workload Analysis for Decentralized Hosting. *Elsevier Computer Networks*, 53(11), 2009.

[82] Shivaram Venkataraman, Niraj Tolia, Parthasarathy Ranganathan, and Roy H. Campbell. Consistent and durable data structures for non-volatile byte-addressable memory. In *USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[83] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, and Bruce Jacob. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News*, 33(4), 2005.

[84] M. Wu and W. Zwaenepoel. eNVy: a Non-Volatile, Main Memory Storage System. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1994.

[85] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid Cache Architecture with Disparate Memory Technologies. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[86] Xilinx. Spartan-6 FPGA Memory Controller User Guide. www.xilinx.com/support/documentation/user_guides/ug388.pdf, 2010.

[87] Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *International Symposium on Circuits and Systems (ISCAS)*, 2007.

[88] Lixin Zhang, Zhen Fang, M. Parker, B.K. Mathew, L. Schaelicke, J.B. Carter, W.C. Hsieh, and S.A. McKee. The Impulse Memory Controller. *IEEE Transactions on Computers, Special Issue on Advances in High-Performance Memory Systems*, 2001.

[89] W. Zhang and T. Li. Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009.

[90] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *International Symposium on Computer Architecture (ISCA)*, 2009.

[91] Y. Zhou, P. Chen, and K. Li. The Multi-Queue Replacement Algorithm for Second-Level Buffer Caches. In *USENIX Annual Technical Conference*, 2001.

Vita

Luiz E. S. Ramos

1998-2002	**B.S. in Computer Science, Pontifical Catholic University of Minas Gerais, Brazil**
2002-2004	**M.S. in Electrical Engineering, Pontifical Catholic University of Minas Gerais, Brazil**
2005-2009	**CAPES/Fulbright Scholarship, Brazil**
2009-2011	**Graduate Assistant, Rutgers University, New Brunswick, New Jersey, USA**
2005-2012	**Ph.D. in Computer Science, Rutgers University, New Brunswick, New Jersey, USA**

Selected Publications

2006	"Mercury and Freon: Temperature Emulation and Management in Server Systems" T. Heath, A. P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. ASPLOS 2006.
2008	"C-Oracle: Predictive Thermal Management for Data Centers" L. Ramos and R. Bianchini. HPCA 2008.
2011	"MemScale: Active Low-Power Modes for Main Memory" Q. Deng, D. Meisner, L. Ramos, T. Wenisch and R. Bianchini. ASPLOS 2011.
2011	"Page Placement in Hybrid Memory Systems" L. Ramos, E. Gorbatov, R. Bianchini. ICS 2011.
2011	"Exploiting Phase-Change Memory in Server Clusters". L. Ramos and R. Bianchini. Technical Report DCS-TR-692, Department of Computer Science, Rutgers University, December 2011.