Exploiting Phase-Change Memory in Cooperative Caches

Luiz Ramos and Ricardo Bianchini
Rutgers University
Piscataway, NJ, USA
{luramos,ricardob}@cs.rutgers.edu

Abstract

Modern servers require large main memories, which so far have been enabled by improvements in DRAM density. However, the scalability of DRAM is approaching its limit, so Phase-Change Memory (PCM) is being considered as an alternative technology. PCM is denser, more scalable, and consumes lower idle power than DRAM, while exhibiting byte-addressability and access times in the nanosecond range. Unfortunately, PCM is also slower than DRAM and has limited endurance. These characteristics prompted the study of hybrid memory systems, combining a small amount of DRAM and a large amount of PCM.

In this paper, we leverage hybrid memories to improve the performance of cooperative memory caches in server clusters. Our approach entails a novel policy that exploits popularity information in placing objects across servers and memory technologies. Our results show that (1) DRAM-only and PCM-only memory systems do not perform well in all cases; and (2) when managed properly, hybrid memories always exhibit the best or close-to-best performance, with significant gains in many cases, without increasing energy consumption.

***Keywords*-Cooperative memory caches; persistent memory.**

I. Introduction

Memory capacity is becoming a scarce resource in server systems, as the number of CPU cores increases, application and thread concurrency escalate, and consolidated virtual machines require co-locating large working sets simultaneously in main memory. Current trends suggest that meeting these 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 [3, 20]. In addition, DRAM is predicted to face scalability issues below the 20nm fabrication process, which would limit the capacity of future DRAM chips [17, 19, 39].

For these reasons, architects have started to consider Phase-Change Memory (PCM) as a potential replacement for DRAM [19, 30, 32, 43, 44]. 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 area. PCM is also expected to scale further than DRAM (beyond 9nm) [17, 33]. However, compared to DRAM, PCM has worse read/write times, read/write energies, and write endurance.

To exploit the low latencies of DRAM and the high capacity of PCM, researchers have proposed hybrid memory systems that combine a small amount of DRAM and a large amount of PCM [30, 32, 43, 44]. However, those approaches (1) are not robust to some workloads (e.g., poor locality or low write-intensity) and/or (2) require hardware modifications. Moreover, hybrid memory systems have not yet been considered for server clusters, such as those of modern Internet services such as Google and Facebook.

Internet services are a great target for hybrid memory systems, since these services require low latency and high memory capacity. In fact, services often organize the server main memories into a single, cluster-wide *cooperative memory cache*. By doing so, they can avoid accessing slow disks or re-generating content. For example, Facebook, Twitter, and Wikipedia are known to use the Memcached cooperative caching middleware [11]. Other proposals, such as the PRESS middleware [6], also implement cooperative caches, but more flexibly than Memcached.

In this paper, we propose Rank-aware Cooperative Caching (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 exploiting PCM's large capacity to increase the hit ratio of the cooperative cache. We apply RaCC to PRESS, in servers with either solid-state drives (SSDs) or hard disk drives (HDDs).

We evaluate these RaCC-based systems using simulation of real and synthetic traces of Internet services. For comparison, we also simulate PRESS with DRAM only, PCM only, and an "unmanaged" system, i.e., a hybrid memory system whose LRU replacement algorithm does not distinguish DRAM from PCM. Our results show that RaCC adds performance robustness to the hybrid coopera-

tive caches. 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. We observe that RaCC's gains depend on the workload locality and the performance of the persistent storage device. To attain robustness, RaCC systems do not consume more energy than the other clusters. For all systems we consider, PCM endurance is not a problem.

We conclude that hybrid memory systems can improve performance and robustness significantly compared to DRAM-only or PCM-only systems. However, for best results, the system needs proper management, as in RaCC.

II. Background

Server memory system. Typically, a server's memory system is composed of a memory controller (MC), a few memory channels, and multiple dual-inline memory modules (DIMMs). Each DIMM includes multiple memory devices (chips), each of which containing a memory cell array and peripheral circuitry. In our technical report [31], we detail these components and the DRAM technology.

PCM. A PCM cell is composed of an access transistor and a storage resistor 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. Via thermal induction, PCM cells can transition between a high electrical resistance state (logical 0) and a low one (logical 1). PCM can interface with most CMOS peripheral circuitry used in DRAM [43]. 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 roughly the same cell size as DRAM. However, PCM enables manufacturing of multi-level cells (MLC), which can store multiple (currently, two or four) bits [26]. Assuming the same cell size for both technologies, these MLCs hold 2x and 4x more data than DRAM cells in the same area [29].

Assumptions for PCM. We assume that, when PCM chips for servers become available, they will be 4x denser than DRAM [30]. 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. Thus, we assume this circuitry to have the same performance and power characteristics for PCM and DRAM [19, 43]. Moreover, only written cache lines in a row buffer are written back to the PCM cell array (DRAM needs the entire buffer to be written back) [19, 44].

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 cell accesses) are slower than with DRAM. Our assumptions for peripheral circuitry imply that row buffer read/write energy is the same for DRAM and PCM.

Cooperative caching. 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 [6, 10, 11, 27]. The middleware directs each object request to the server likely to be caching the object.

In this paper, we study PRESS [6], which does 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 (hundreds of nanoseconds) across a high-performance switch (e.g., InfiniBand) is faster than accessing a local HDD (several milliseconds) or a local SSD (dozens of microseconds).

PRESS implements request forwarding and load balancing using a Global Directory (GD) structure, replicated in every server. Each server updates the GD in two situations: (1) when it starts or stops caching an object; and (2) periodically, to inform the other servers about its load level (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 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. PRESS limits the size of the largest cacheable object.

III. The RaCC Policy

In this section, we present the RaCC policy. RaCC is an object placement policy for cooperative caches where both DRAM and PCM comprise the memory of each server. Previous research has shown that typically only a relatively small subset of objects are very frequently accessed in Web caches [12, 36]. Moreover, that subset changes over time. These observations suggest 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.

A. Basic mechanisms

Object ranking. RaCC tracks the popularity of objects stored in the cooperative cache, and uses that knowledge in

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

Algorithm 1: RaCC's local placement algorithm.

placing them. Specifically, RaCC tracks the frequency and recency of references to the cached objects. To dynamically rank objects, each server builds two private Multi-Queue (MQ) structures [45] locally: PMQ and DMQ, updated as requests for content arrive at the servers. *PMQ* ranks the objects stored solely in PCM, and *DMQ* ranks those stored solely in DRAM. No object spans both memory regions.

Each MQ structure stores object descriptors into M LRU queues, numbered from 0 to $M-1$. Each descriptor holds the object number, a reference counter (RC), and a logical expiration time (LET). We measure logical time in number of accesses to objects, and accumulate the total into *CurrentTime* (CT). On the first access to an object, MQ places its descriptor in the tail of queue 0 and sets its LET to $CT+\lambda$, where λ is a constant. The object expires if not accessed until time $CT+\lambda+1$. On every new access to the object, we increment its RC, reset its LET to $CT+\lambda$, and move its descriptor to the tail of its current queue. If the descriptor is currently in queue i, it will be upgraded to queue $i+1$ (if $i+1<M$) when its RC reaches 2^{i+1}. Conversely, on each access, we check the descriptors at the heads of all M queues for expiration ($CT>LET$). We place an expired descriptor at the tail of the immediately inferior queue, and set its LET to $CT+\lambda$. When an object cached in PCM reaches the queue ρ of PMQ, it is considered popular. In our simulations, we set M to 16, λ to 32, and ρ to 8, as these values showed good empirical results.

Server-level object placement. RaCC manages memory as shown in Algorithm 1. RaCC first tries to allocate objects 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 (DRAM or PCM) has free space, the replacement algorithm evicts as many unpopular objects as necessary to make room for the new object (function *ensureRoom*). However, because RaCC will cache the new

```
1  function coopCacheServe (Request r) begin
2      Object obj = r.target
3      if obj.size > MAX_SIZE then
4          replyToClient (r.client, serveFromDisk (r))
5      else if obj is not cached on any server then
6          startCaching (obj)
7          replyToClient (r.client, serveFromCache (obj))
8      else if obj is cached on the initial server then
9          replyToClient (r.client, serveFromCache (obj))
10         tryMigrate (obj)
11     else
12         balanceLoad (r, obj)
13 function balanceLoad (Request r, Object obj) begin
14     PeerServer w = least loaded server caching obj in DRAM
15     if GD.notOverloaded (w) then
16         forwardToPeer (r, w)
17         replyToClient (r.client, receiveReplyFromPeer (w))
18         return
19     PeerServer z = least loaded server caching obj in PCM
20     if GD.notOverloaded (z) then
21         forwardToPeer (r, z)
22         replyToClient (r.client, receiveReplyFromPeer(z))
23         return
24     if GD.notOverloaded (localhost) then
25         startCaching (obj)
26         replyToClient (r.client, serveFromCache(obj))
27         return
28     PeerServer v = least loaded server in the cluster
29     if GD.notOverloaded (v) then
30         forwardToPeer (r, v)
31         replyToClient (r.client, receiveReplyFromPeer(v))
32     else
33         forwardToPeer (r, w)
34         replyToClient (r.client, receiveReplyFromPeer(w))
35 function receiveForwardedRequest (Request r, PeerServer p) begin
36     if r.target is not cached then startCaching (r.target)
37     replyToPeer (p, serveFromCache (r.target))
38     tryMigrate (obj)
```

Algorithm 2: RaCC for PRESS. The highlighted lines were modified or added to PRESS.

object in a single memory region, all unpopular objects must be evicted either from DRAM or from PCM (note that objects in DRAM may become unpopular over time, thus becoming potential victims for popular objects in PCM.) To select the victim region, RaCC finds the set of least popular objects in both DMQ and PMQ (lines 9-10). Within each group of unpopular objects, RaCC finds a "pivot" (the most popular object), then it compares the popularity of both pivots (line 11). If the pivots occupy different queues in DMQ and PMQ, RaCC selects the region with the lowest-ranked pivot. Otherwise, RaCC selects the region with the LRU pivot.

B. 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 2 depicts the PRESS behavior

upon receiving a request, when augmented with RaCC. The modifications introduced by RaCC (highlighted) are: the search for opportunities to migrate popular objects in DRAM (lines 10 and 38); and the forwarding of requests preferably to servers already caching the requested object in DRAM (lines 14-23).

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 – 1MByte in our simulations (lines 3-4); (2) seen for the first time (lines 5-7); or (3) already cached locally in DRAM or PCM (lines 8-9). Otherwise, the initial server tries to forward the request to another server while balancing load (lines 11-12). 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 14-18). If that server is not found, RaCC looks for a server caching the object in PCM (lines 19-23). 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 24-27). 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 28-31). If none of the previous cases is satisfied, RaCC forwards the request to the server that has the object in DRAM (lines 33-34), despite overloading. Upon receiving a forwarded request, a server may need to update its cache, and then send the cached object to the initial server (lines 36-37). Whenever a server starts caching the requested object, it broadcasts that fact to all servers, updating the GD, just like PRESS. 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. If the object reaches queue ρ 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 2 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 occurs, 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 added to PRESS.

Table I. 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%

Object replication and popularity hinting. The 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 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 a 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 1 to place the replica. Over time, RaCC concentrates popular objects in DRAM.

GD modifications. 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* (OSEs), which represent the servers caching the object locally. When a server starts caching an object or replica, it creates a new OSE in the object map of the GD. An OSE contains a pointer to an array (indexed by server ID) that holds data about the server's load level.

We extend each OSE with a single-bit flag, which indicates whether an object resides in the server's DRAM. In addition, when RaCC migrates an object into the DRAM of a server, the server broadcasts that fact, and all servers set the corresponding flag in their copy of the GD.

IV. Evaluation Methodology

Workloads. We use real and synthetic HTTP traces from different sources. *Wiki* samples 2% of a real Wikipedia trace in all languages (November'07) [38]. *Ircache* contains actual requests from two medium/large Web caches (January'07) [37]. *Govcomm* combines several real traces from the 1990's (ClarkNet, EPA, NASA, and World Cup'98). We generated *twitter* and *flickr* synthetically using scripts and object popularity distributions provided by researchers at the University of Michigan. The traces follow empirically measured and fitted distributions, respectively, from Twitter.com and Flickr.com. Table I summarizes the main characteristics of our workloads: (1) the average and peak offered loads (in requests per second), (2) the average size of requests in KB, (3) the data sizes in

Table II. Cluster node features

Component	Values and units
CPU request forward / migration (empirically)	500ns / 1μs
CPU idle / max power [14]	30W / 105W
SSD 4KB read [18]	25μs
SSD idle / max power [18]	0.75W / 6W
HDD 4KB read [35]	2.4ms
HDD idle / max power [35]	4W / 7W
Infiniband latency [22] / 1KB transfer [13]	100ns / 80ns
Infiniband idle / max power [21]	11W / 13W
DDR3-1866 [19, 23, 29]	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

GBytes and in number of objects, and (4) the average hit ratio when the system is solely composed of either DRAM or PCM (determined by cache sizes and workload locality). Our report shows the distribution of object popularity for each workload [31].

Because the workloads have small and varied working set sizes, we provisioned the server memories according to those sizes. For each workload, we found (via simulation) the smallest memory for which a DRAM-only system serves requests at hundreds of nanoseconds (as in our validation experiments). Since no single size met this criterion for all workloads, we set our baseline DRAM DIMM size to 32MBytes for govcomm and ircache, 192MBytes for twitter and flickr, and 1GBytes for wiki. We assume 2 DIMMs per server (for hybrid systems: 1 DRAM + 1 PCM DIMM). The resulting cache sizes per server for DRAM-only, hybrid memory systems, and PCM-only are respectively: 64MBytes, 160MBytes, or 256MBytes (for govcomm and ircache); 384MBytes, 960MBytes, or 1536 MBytes (for twitter and flickr); and 2GBytes, 5GBytes, or 8GBytes (for wiki).

Simulation infrastructure. Because real PCM hardware is not yet widely available, we use simulation in our evaluation. We simulate servers that receive 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 a public network, and to each other via a private network.

Our event-driven simulator replays HTTP request traces in open-loop. The simulator models each server as a collection of components (CPU, SSD/HDD, NICs, and memory), which are modeled using: a service rate, a utilization level, and a wait queue to store requests until it becomes available. Processing different requests may entail different component-visiting sequences (e.g., a cache miss causes a visit to the disk, but a cache hit does not). We keep track of object location in memory and count writes to PCM frames for endurance calculations. We estimate the load of a server by looking at its amount of pending requests (e.g., for overload detection in PRESS). Our validation against a real server found real DRAM response times to be within

Table III. RBHR influence on memory accesses [19, 29]

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

14% of our simulations [31]. To derive the cluster's energy consumption, we compute the servers' average power per simulated second. The power is composed of the idle power (constant) and the dynamic power (a linear function of the simulated utilization) of all server components. We compute memory energy as in [19].

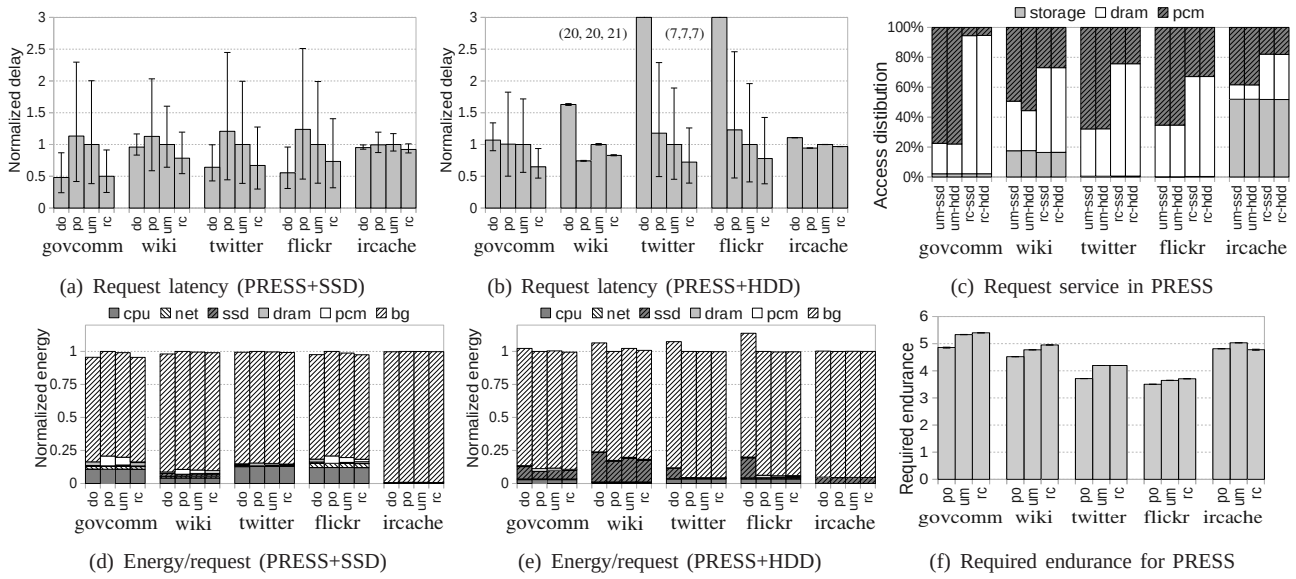
We simulate 8 servers, each with an 8-core CPU, DDR3-1866 memory DIMMs, SSD or HDD storage, 1Gb/sec Ethernet NIC (public network), and a 4X EDR Infiniband NIC (private network). Table II summarizes the power and timing characteristics of our servers. Our DDR3 and DRAM power and timing parameters are from [23, 40], and PCM parameters from [19, 29].

To study a range of object-to-memory mappings, we parametrize the simulator with the expected row buffer hit ratio (RBHR) [23]. A low RBHR brings out the performance and energy of the memory devices in each access, whereas a high RBHR brings the performance and energy of DRAM and PCM closer to each other, as we show in Table III. Because bank interleaving reduces the row buffer locality (in exchange for better throughput) and the physical frames of the requested objects are unlikely to be contiguous, we expect a maximum RBHR of 12.5% or lower, due to conflicts between DMA streams. We analyze the sensitivity of our results to the RBHR.

Baselines for comparison. Because RaCC has been designed for clusters with hybrid memory, we compare it to an *Unmanaged* hybrid memory cluster, whose LRU replacement policy does not explicitly differentiate between DRAM and PCM. We also compare the hybrid memory systems to two baselines: DRAM-only and PCM-only clusters, running the original version of PRESS. We also consider the impact of the persistent storage devices (SSD and HDD) on the service performance. Recent studies suggest that SSDs are not a cost-effective replacement for HDDs in datacenters, although their combination can be beneficial [25]. In such a scenario, performance should lie between the SSD and HDD results we present.

V. Performance Evaluation

Figure 1(a) shows the latency per request served by a PRESS cluster, where each server uses an SSD for persistent storage. The bars 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



(a) Request latency (PRESS+SSD) (b) Request latency (PRESS+HDD) (c) Request service in PRESS

(d) Energy/request (PRESS+SSD) (e) Energy/request (PRESS+HDD) (f) Required endurance for PRESS

Figure 1. (a,b) Average latency per served request, normalized to Unmanaged. (c) Percentage of requests served from each memory and storage device. (d,e) Energy per request (total enegy divided by the number of requests served). (f) PCM's required endurance over a period of 5 years in logarithmic scale. Do = DRAM-only, po = PCM-only, um = Unmanaged, and rc = RaCC.

and the lowest latencies across RBHRs, respectively. In general, as 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. High RBHRs hide the performance disadvantage of PCM. In fact, all maximum latencies 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 I), 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 for slower SSDs). 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 two 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 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. Compared to Unmanaged, RaCC performs 28% better, on average. The reason is that RaCC concentrates popular objects in DRAM, creating more opportunities to serve them from that region. We show this in Figure 1(c), which quantifies the percentage of requests that the cluster serves from its SSDs (on cache misses), DRAM, and PCM. The figure shows that 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 1(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. 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 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 1(c), RaCC is able to serve more requests from DRAM than Unmanaged, thus presenting better average performance. Using HDDs, 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 III-B), 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.

VI. Energy Evaluation

In this section, we study the static (background or bg) 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.

Figure 1(d) shows the breakdown of energy consumption for SSD-based clusters. The static energy of the servers in periods of low and high utilization adds up and dominates the total energy consumption. We do not turn off servers to conserve energy, since the cached data would be lost (if in DRAM) or become stale (if in PCM). Overall, the total energy per request is similar across memory organizations, all within 2% of each other. In fact, the dynamic energy of RaCC is always close to that of DRAM-only (within 3%), which exhibits the best energy consumption per request. 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. By concentrating most accesses in DRAM, RaCC exhibits a better average dynamic energy than Unmanaged (9%) and PCM-only (13%).

Figure 1(e) shows the breakdown of the energy consumption for HDD-based clusters. Like in the SSD case, we note that the static energy is 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 HDDs dominates the total energy per request. In fact, RaCC's total energy consumption lies within 1% of Unmanaged and PCM-only. Because of the high penalty of cache misses, DRAM-only is slightly worse than RaCC by 6%. 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%).

VII. Endurance Evaluation

In this section, we compare the endurance of PCM in the systems we study, using the Required Endurance metric [5]: $T_{life} \times \frac{B}{\alpha\ C}$, 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 (the lower, the worse), and C is the memory capacity in bytes. For each workload, we consider the typical 5-year server lifespan as T_{life}. α is A/M, where A is the average number of writes per PCM frame and M is the number of writes to the most written frame in the PCM area of the system. Required Endurance determines the number of writes that a PCM frame must be able to withstand during the server's lifespan.

Figure 1(f) compares 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 $10^8 - 10^9$. We notice that, all systems exhibit relatively low PCM endurance requirements, due to their limited memory write throughput and large PCM capacity. RaCC requires roughly the same endurance as Unmanaged (within 1% of each other). Because PCM-only has larger capacity on average, it requires 7% less PCM endurance than both hybrid systems. With larger capacity, the PCM-only cooperative cache exhibits higher hit ratio and lower write traffic to PCM than the hybrid systems.

These results show that PCM endurance is not a problem for the clusters and workloads we consider. However, we could combine RaCC with current techniques for improving PCM endurance, e.g. [9, 15, 19, 44]. Moreover, the predictions for future PCM cells suggest significant endurance improvement (10^{15} writes by 2022 [16]), whereas PCM's performance and energy will still likely lag those of DRAM. Thus, we focus on the latter two aspects.

VIII. Related Work

Cooperative caching. Many locality-aware cooperative caches have been proposed in the literature [1, 2, 4, 6, 8, 11, 27]. However, our paper 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 differences.

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 [24, 41]. With better characteristics than Flash, PCM is a more promising technology for use in main memory. In fact, several recent works [7, 9, 19, 30, 28, 32, 42–44] have proposed systems that use PCM as a partial or complete DRAM replacement.

For at least three reasons, RaCC differs from previous approaches that manage data placement in flat hybrid sys-

tems. 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. Although hardware approaches may behave similarly in some cases, they add complexity to the memory controller and consume extra on-chip area. Manufacturers could, therefore, sell software-manageable PCM at lower cost.

Tackling PCM's endurance problem. Many previous works focused extensively on the work-arounds needed to mitigate this problem [9, 15, 19, 30, 28, 34, 42, 44]. Our design for RaCC is orthogonal to these techniques. In fact, it can be combined with one or more of them to improve PCM's lifetime, hence, we aim at improving PCM's performance without increasing energy consumption.

IX. Conclusions

We introduced RaCC, an object placement policy for server clusters that use hybrid DRAM+PCM main memories in cooperative caches. RaCC ranks, migrates, and replaces objects to improve the cache's performance. Our results show that the combination of hybrid memories and intelligent object placement can produce efficient and robust clusters without increasing energy consumption.

Acknowledgments. We thank David Meisner and Thomas Wenisch for the Twitter and Flickr data, and our sponsors: NSF (grant #CCF-0916539), Intel, and CAPES (Brazil).

References

[1] V. Anagnostopoulou et al. Barely Alive Memory Servers: Keeping Data Active in a Low-Power State. *ACM JETC*, 2012.
[2] D. G. Andersen et al. FAWN: A Fast Array of Wimpy Nodes. In *SOSP*, 2009.
[3] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to The Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.
[4] R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based Network Servers. *WWW Journal*, 3(4), 2000.
[5] G. W. Burr et al. Phase-Change Memory Technology. *Journal of Vacuum Science & Technology B*, 2010.
[6] E. V. Carrera and R. Bianchini. PRESS: A Clustered Server Based on User-Level Communication. *TPDS*, 16, May 2005.
[7] A. Caulfield et al. Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing. In *SC*, 2010.
[8] D. Chiu, A. Shetty, and G. Agrawal. Elastic Cloud Caches for Accelerating Service-Oriented Computations. In *SC*, 2010.
[9] S. Cho and H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *MICRO*, 2009.
[10] G. DeCandia et al. Dynamo: Amazon's Highly Available Key-Value Store. In *ACM SIGOPS*, 2007.
[11] B. Fitzpatrick. Distributed Caching With Memcached. In *Linux Journal*, 2004.
[12] L. Guo et al. Does Internet Media Traffic Really Follow Zipf-Like Distribution? In *SIGMETRICS*, 2007.
[13] InfiniBand Trade Association. InfiniBand Roadmap, 2011. www.infinibandta.org/content/pages.php? pg=technology_overview.
[14] Intel. Xeon E7-8830, 2011. ark.intel.com/products/53677/.
[15] E. Ipek et al. Dynamically Replicated Memory: Building Reliable Systems From Nanoscale Resistive Memories. In *ASPLOS*, 2010.
[16] ITRS. Emerging Research Devices. www.itrs.net/Links/2007ITRS/2007_Chapters/2007_ERD.pdf, 2007.
[17] ITRS. Emerging Research Devices, 2009. www.itrs.net/links/2009itrs/2009chapters_2009tables/2009_PIDS.pdf.
[18] A. Ku. Meet Intel's SSD 320. www.tomshardware.com/reviews/intel-ssd-320-crucial-m4-realssd-c400,2908-2.html, 2011.
[19] B. Lee et al. Architecting Phase-change Memory as a Scalable DRAM Architecture. In *ISCA*, 2009.
[20] C. Lefurgy et al. Energy Management for Commercial Servers. *IEEE Computer*, 36(12), December 2003.
[21] Mellanox Technologies. Adapter Card Product Guide, 2009. www.colfaxdirect.com/store/pc/catalog/DDR_Bundle1.pdf.
[22] Mellanox Technologies. Voltaire Grid Director 4036E, 2011. www.voltaire.com/Products/InfiniBand/Grid_Director_Switches/voltaire_grid_director_4036E.
[23] Micron. 1Gb:x4, x8, x16 DDR3 SDRAM Features. 2006.
[24] J. C. Mogul et al. Operating System Support for NVM+DRAM Hybrid Main Memory. In *HotOS*, 2009.
[25] D. Narayanan et al. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *EuroSys*, 2009.
[26] T. Nirschl et al. Write Strategies for 2 and 4-Bit Multi-Level Phase-Change Memory. In *IEDM*, 2007.
[27] V. Pai et al. Locality-Aware Request Distribution in Cluster-Based Network Servers. *ASPLOS*, 1998.
[28] M. K. Qureshi et al. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. In *MICRO*, 2009.
[29] M. K. Qureshi et al. Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase-Change Memories. In *ISCA*, 2010.
[30] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *ISCA*, 2009.
[31] L. Ramos and R. Bianchini. Exploiting Phase-Change Memory in Server Clusters, 2011. Tech. Rep. DCS-TR-692, Rutgers University.
[32] L. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. In *ICS*, 2011.
[33] S. Raoux et al. Phase-Change Random Access Memory: A Scalable Technology. *IBM Journal of Research and Development*, 2008.
[34] S. Schechter, G. H. Loh, K. Straus, and D. Burger. Use ECP, Not ECC, for Hard Failures in Resistive Memories. In *ISCA*, 2010.
[35] Seagate. Savvio 15K.2 Data Sheet, 2010. www.seagate.com/docs/pdf/datasheet/disc/ds_savvio_15k_2.pdf.
[36] L. Shi et al. An Applicative Study of Zipf's Law on Web Cache. *International Journal of Information Technology*, 12(4), 2006.
[37] The NLANR Web Caching Project. www.ircache.net. 2010.
[38] G. Urdaneta et al. Wikipedia Workload Analysis for Decentralized Hosting. *Elsevier Computer Networks*, 53(11), 2009.
[39] S. Venkataraman et al. Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory. In *FAST*, 2011.
[40] D. Wang et al. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News*, 33(4), 2005.
[41] M. Wu and W. Zwaenepoel. eNVy: a Non-Volatile, Main Memory Storage System. In *ASPLOS*, 1994.
[42] B.-D. Yang et al. A Low Power Phase-Change Random Access Memory Using a Data-Comparison Write Scheme. In *ISCAS*, 2007.
[43] W. Zhang and T. Li. Exploring Phase-Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In *PACT*, 2009.
[44] P. Zhou et al. A Durable and Energy Efficient Main Memory Using Phase-Change Memory Technology. In *ISCA*, 2009.
[45] Y. Zhou, P. Chen, and K. Li. The Multi-Queue Replacement Algorithm for Second-Level Buffer Caches. In *USENIX*, 2001.