International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 14
October2016

Hybrid Memory Systems to Improve the Performance of PCM

C Mariveedu Suresh [1] & K Madanna [2]

[1]M-Tech Dept of ECE, Geethanjali Engineering College NANNUR-V, KURNOOL-DIST

[2] Assistant Professor Dept ECE, Geethanjali Engineering College NANNUR-V, KURNOOL DIST Mail Id:- madanna2011@gmail.com

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, Phase-Change Memory (PCM).

1. INTRODUCTION

The concept of using the amorphous to crystalline phase transition of chalcogenides for an electronic memory technology has been pursued for many years [1]–[2]. While the early work disclosed many of the fundamental concepts of the phase change memory (PCM), it is only in the past 10–15 years that advances in materials and device technology have made it possible to demonstrate PCMs that rival incumbent technologies such as Flash [3]. The characteristics of PCM most closely approximate that of the dynamic random access memory (DRAM) and the Flash memory [4]. Reports on PCM have grown rapidly in recent years. The worldwide research and development effort on emerging memory devices and PCM in particular can be understood from two perspectives. First, from a system point of view, processor performance is increasingly limited by memory access and power consumption of the memory subsystem. Recent efforts in extending the scalability of SRAM and incorporating embedded DRAM in advanced technologies are evidence of the importance of the memory technology. The emergence of Flash as a potential solid-state replacement for the hard disk drive (HDD) for selected applications has highlighted the enormous potential of a high-density, embedded memory technology within the memory hierarchy. At the same time, memory device research has had a renaissance of new ideas [5], [6]. New memory devices, most of them nonvolatile, have been explored and some have progressed beyond the observation of a hysteresis effect to device-level demonstrations. These new memory devices, such as PCM, have read/write/retention/endurance characteristics different from conventional static random access memory (SRAM), DRAM, and Flash. The very high density offered by some of the new device technologies may also lead to the replacement of the HDD by solid-state devices for some applications [7], [8]. There is an enormous opportunity to completely rethink the design of the memory subsystem to gain orders of magnitude improvements in speed and/or power consumption. A revolution in the memory subsystem will bring about a fundamental change in how one can extract performance out of technology improvements.

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. In addition, DRAM is predicted to face scalability issues below the 20nm fabrication process, which would limit the capacity of future DRAM chips. For these reasons, architects have started to consider Phase-Change Memory (PCM) as a potential replacement for DRAM. 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) [10, 9]. However, compared to DRAM, PCM has worse read/write times, read/write energies, and write endurance.

2. RELETED WORK

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 [11], 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 [12]. 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.

Assumptions for PCM: We assume that, when PCM chips for servers become available, they will be 4x denser than DRAM. 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. 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) [13, 14].

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. The middleware directs each object request to the server likely to be caching the object.

Tackling PCM's endurance problem: Many previous works focused extensively on the work arounds needed to mitigate this problem. 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.

3. IMPLEMENTATION

Object ranking: RaCC tracks the popularity of objects stored in the cooperative cache, and uses that knowledge in 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 [15] 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.

```
function startCaching (Object obj) begin
    if DRAM.hasEnoughFreeFrames(obj.size) then
        cacheObject (obj, DRAM)
        DMQ.add (obj)
    else if PCM.hasEnoughFreeFrames(obj.size) then
        cacheObject (obj, PCM)
        PMQ.add (obj)
    else
        DMQpivot = DMQ.getLeastPopularInSet(obj.size)
        PMQpivot = PMQ.getLeastPopularInSet(obj.size)
        if DMQpivot is less popular than PMQpivot then
            ensureRoom (obj.size, DRAM, DMQ)
            cacheObject (obj, DRAM)
            DMQ.add (obj)
        else
            ensureRoom (obj.size, PCM, PMQ)
            cacheObject (obj, PCM)
            PMQ.add (obj)
```

Algorithm 1: RaCC's local placement algorithm.

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+λ, where λ is a constant. The object expires if not accessed until time CT+λ + 1. On every new access to the object, we increment its RC, reset its LET to CT+λ 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 2i+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+λ. 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 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.

```
function coopCacheServe (Request r) begin
    Object obj = r.target
    if obj.size > MAX_SIZE then
        replyToClient (r.client, serveFromDisk (
    else if obj is not cached on any server then
        startCaching (obj)
        replyToClient (r.client, serveFromCache
    else if obj is cached on the initial server the
        replyToClient (r.client, serveFromCache
    else
        balanceLoad (r, obj)

function balanceLoad (Request r, Object obj) beg
    if GD.notOverloaded (localhost) then
        startCaching (obj)
        replyToClient (r.client, serveFromCache(
        return
    PeerServer v = least loaded server in the clus
    if GD.notOverloaded (v) then
        forwardToPeer (r, v)
        replyToClient (r.client, receiveReplyFror
    else
        forwardToPeer (r, w)
        replyToClient (r.client, receiveReplyFror
```

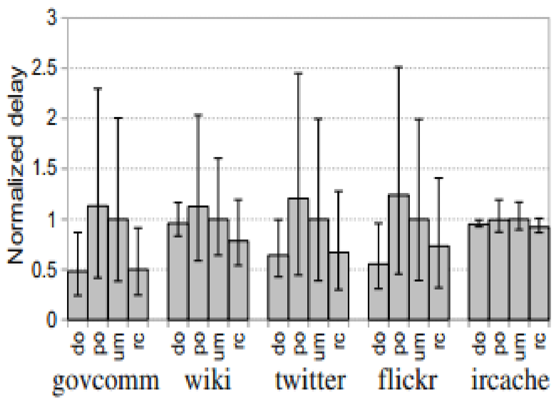
Algorithm 2: RaCC for PRESS. The highlighted lines were modified or added to PRESS.

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, and the forwarding of requests preferably to servers already caching the requested object in DRAM.

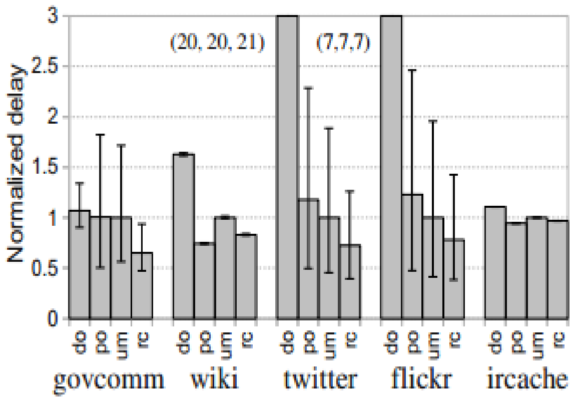
HTTP requests arrive via standard round-robin DNS to any server. The server that initially receives the request (called “initial server”) parses it, inspects its content, and decides where it should be served. The initial server itself serves requests for objects that are (1) larger than the maximum cacheable size – 1MByte in our simulations seen (2) for the first time, or (3) already cached locally in DRAM or PCM. Otherwise, the initial server tries to forward the request to another server while balancing load. 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. If that server is not found, RaCC looks for a server caching the object in PCM. 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. 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. If none of the previous cases is satisfied, RaCC forwards the request to the server that has the object in DRAM, 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. 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.

4. EXPERIMENTAL RESULTS

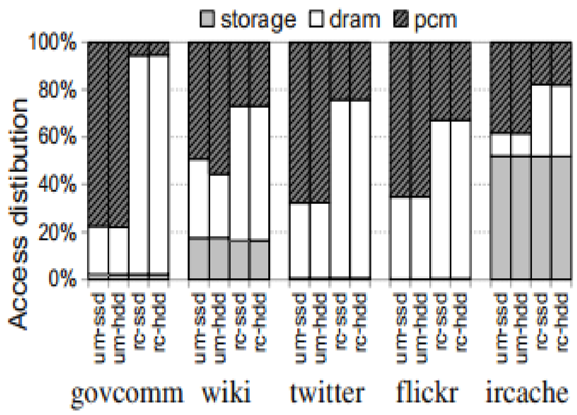
Figure 1(a) shows the latency per request served by a PRESS cluster, where each server uses an SSD for persistent storage. The bars represent the average latency per request across different RBHRs (1%, 6%, and 12%). The upper and lower ends of the error bars represent the highest and the lowest latencies 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.



(a) Request latency (PRESS+SSD)



(b) Request latency (PRESS+HDD)



(c) Request service in PRESS

Figure 1. (a,b) Average latency per served request, normalized to Unmanaged. (c) Percentage of requests served from each memory and storage device.

Figure 1(a,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, DRAMonly 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, 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.

5. CONCLUSION

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.

6. REFERENCES

[1] S. R. Ovshinsky, BSymmetrical current controlling device,[U.S. Patent 3 271 591,1966

[2] S. R. Ovshinsky, BReversible electricalswitching phenomena in disordered

structures,[Phys. Rev. Lett., vol. 21,pp. 1450–1453, 1968.

[3] S. Lai, BCurrent status of the phase change memory and its future,[in Proc. IEEE Int. Electron Devices Meeting, 2003,pp. 10.1.1–10.1.4.

[4] International Technology Roadmap for Semiconductors (ITRS), 2009. [Online]. Available: http://public.itrs/net Available: http://public.itrs/net

[5] K. Kim, BFuture memory technology:Challenges and opportunities,[in Proc. Symp. Very Large Scale Integr. Technol.Syst. Appl., 2008, pp. 5–9.

[6] K. Galatsis, K. Wang, Y. Botros, Y. Yang,Y.-H. Xie, J. F. Stoddart, R. B. Kaner, C. Ozkan, J. Liu, M. Ozkan, C. Zhou, and K. W. Kim, BEmerging memory devices,[IEEE Circuits Devices Mag., vol. 22, no. 3,pp. 12–21, May/Jun. 2006.

[7] G. W. Burr, B. N. Kurdi, J. C. Scott,C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, BOverview of candidate device technologies for storage-class memory,[IBM J. Res. Develop.,

vol. 52, no. 4, pp. 449–464, 2008.

[8] M. H. Kryder and C. S. Kinm, BAfter hard drivesVWhat comes next?'' IEEE Trans. Magn., vol. 45, no. 10, pp. 3406–3413,Oct. 2009.

[9] InfiniBand Trade Association. InfiniBand Roadmap, 2011. www.infinibandta.org/content/pages.php?pg=technology overview.

[10] ITRS. Emerging Research Devices, 2009. www.itrs.net/links/2009itrs/2009chapters 2009tables/2009 PIDS.pdf.

[11] L. Ramos and R. Bianchini. Exploiting Phase-Change Memory in Server Clusters, 2011. Tech. Rep. DCS-TR-692, Rutgers University.

[12] W. Zhang and T. Li. Exploring Phase-Change Memory and 3D DieStacking for Power/Thermal Friendly, Fast and Durable Memory Architectures. In PACT, 2009.

[13] B. Lee et al. Architecting Phase-change Memory as a Scalable DRAM Architecture. In ISCA, 2009.

[14] P. Zhou et al. A Durable and Energy Efficient Main Memory Using Phase-Change Memory Technology. In ISCA, 2009.

[15] Y. Zhou, P. Chen, and K. Li. The Multi-Queue Replacement Algorithm for

Second-Level Buffer Caches. In USENIX, 2001.