Mortar: Filling the Gaps in Data Center Memory

Jinho Hwang, Ahsen J. Uppal, Timothy Wood, H. Howie Huang
The George Washington University

1 Introduction

Data center servers are typically overprovisioned, leaving spare memory and CPU capacity idle to handle unpredictable workload bursts by the virtual machines running on them [1, 2, 3]. While this allows for fast hotspot mitigation, it is also wasteful. Unfortunately, making use of spare capacity without impacting active applications is particularly difficult for memory since it typically must be allocated in coarse chunks over long timescales [4, 5, 6, 7]. In this work we propose repurposing the poorly utilized memory in a data center to store a volatile data store that is managed by the hypervisor. We present two uses for our Mortar framework: as a cache for prefetching disk blocks [8, 9, 10], and as an application-level distributed cache that follows the memcached protocol [11, 12]. Both prototypes use the framework to ask the hypervisor to store useful, but recoverable data within its free memory pool. This allows the hypervisor to control eviction policies and prioritize access to the cache.

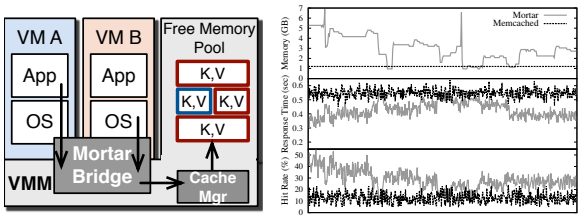
The contributions of this paper are as follows:

- A framework for repurposing spare system memory that otherwise would be idle or poorly utilized.
- A prototype disk prefetching system that aggressively reads disk blocks into spare hypervisor memory to reduce the latency of future disk reads.
- An enhanced memcached server that can utilize hypervisor controlled memory to build a distributed app-level cache accessible by web applications.
- Cache allocation and replacement algorithms for prioritizing access to spare memory and balancing the need to retain hot data in the cache against the goal to immediately reclaim memory for other uses.

SoCC'13, 1–3 Oct. 2013, Santa Clara, California, USA.
ACM 978-1-4503-2428-1.
http://dx.doi.org/10.1145/2523616.2525937

2 Mortar Overview

The Mortar framework is divided into two main components as shown in Figure 1(a). The Mortar Bridge is composed of a pair of interfaces at the hypervisor and kernel levels that allow user applications to transfer data to and from the hypervisor's free memory pool. This interface can be accessed via system calls within user-space, or with direct hypercalls in kernel-space. The request to put or retrieve data from the hypervisor is passed to the Mortar Cache Manager, which is responsible for managing the hypervisor's free memory pool.



(a) Mortar Architecture (b) Memory Trace and Performance

Figure 1: (a) Mortar allows an application or OS to store Key-Value data in the hypervisor's free memory pool; (b) Mortar makes use of all spare memory, leading to lower response times and a higher hit rate than memcached.

We present two example usages:

- Mortar-based Memcached: aggregates free memory throughout the data center for use as a distributed cache following the standard memcached protocol. This allows unmodified web applications to achieve performance gains by opportunistically using spare data center memory.
- Mortar-based Prefetching: can be used at the OS-level to transparently cache and prefetch disk blocks for applications. Prefetching is an ideal candidate for Mortar's volatile data store because the aggressiveness of the algorithm can be tuned based on the amount of free memory available.

Figure 1(b) illustrates the allotted memory (top) for each case, the response time (middle), and the hit rate (bottom) as web requests are processed over the course of the experiment. Mortar has an average response time of 0.38 seconds, a 35% improvement over memcached.

References

[1] Carl A. Waldspurger. Memory resource management in vmware esx server. *OSDI*, 2002.

[2] Luiz Andre Barroso and Urs Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 2009.

[3] Dan Magenheimer, Chris Mason, Dave McCracken, and Kurt Hackel. Transcendent memory and linux. *Oracle Corp.*, 2009.

[4] VMware. Resource management with vmware drs. *Technical Resource Center*, 2006.

[5] Y. Joo, J. Ryu, S. Park, and K.G. Shin. FAST: quick application launch on solid-state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, pages 19–19. USENIX Association, 2011.

[6] Weiming Zhao and Zhenlin Wang. Dynamic memory balancing for virtual machines. *VEE*, 2009.

[7] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Saving cache by using less cache. *HotCloud*, 2012.

[8] B. S. Gill and D. S. Modha. SARC: sequential prefetching in adaptive replacement cache. In *Proceedings of the USENIX Annual Technical Conference*. Berkeley, CA, USA, 2005.

[9] Xiaoning Ding, Song Jiang, Feng Chen, Kei Davis, and Xiaodong Zhang. Diskseen: exploiting disk layout and access history to enhance i/o prefetch. In *USENIX Annual Technical Conference*, pages 20:1–20:14, 2007.

[10] A.J. Uppal, R.C. Chiang, and H.H. Huang. Flashy prefetching for high-performance flash drives. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1 –12, april 2012.

[11] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling memcache at facebook. *USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[12] Jinho Hwang and Timothy Wood. Adaptive performance-aware distributed memory caching. *USENIX Internation Conference on Autonomic Computing*, 2013.