Rethinking Software Runtimes for Disaggregated Memory
Extended Abstract

Irina Calciu[1], Talha Imran[2], Ivan Puddu[3], Sanidhya Kashyap[4], Hasan Maruf[5], Onur Mutlu[3], Aasheesh Kolli[2]
[1]*VMware Research,* [2]*Penn State University,* [3]*ETH Zürich,* [4]*EPFL,* [5]*University of Michigan*

1. Motivation

Disaggregated memory addresses resource provisioning inefficiencies in current datacenters by improving memory utilization and decreasing the total memory over-provisioning necessary to avoid out-of-memory errors or swapping [3]. In addition, disaggregated memory enables independent scaling of memory and compute, and it disentangles hardware failures and replacements from the monolithic server. Fine-grain microsecond-latency networking technologies, such as Remote Direct Memory Access (RDMA) [8, 14, 15] and Gen-Z [6] are key enablers for hardware disaggregated memory and make the technology feasible in the near future.

Unfortunately, enabling applications to efficiently adopt disaggregated memory is not straightforward. Many software systems [10, 4, 2, 7, 12] have been proposed to enable applications to *transparently*, without code changes, use remote memory – the memory of another host in the rack or memory that has been physically disaggregated from the compute. These systems use various kernel subsystems [7, 2, 12] or redesign the kernel altogether [11]. Fundamentally, they all rely on the core virtual memory mechanism for three essential functions: 1) *fetching remote data* by detecting remote accesses using page faults, and *caching* the remote pages in a local DRAM cache; 2) *dirty data tracking* the cached data by write-protecting pages and causing page faults on the first write of each page; and 3) *evicting cached pages* from the local DRAM cache, which requires marking the pages as not present and flushing the translation look-aside buffers (TLBs).

Virtual memory provides application transparency, but results in high overhead and causes a significant drop in application performance, even when the amount of remote data accessed is small. Page faults latencies exceed network latencies, creating a bottleneck in the system software stack. Moreover, virtual memory requires moving and tracking data at page-granularity, with a page size of 4KB or higher. In contrast, throughout their lifetimes, applications often access only a small part of each page, causing large *amplification* and poor network utilization, by re-writing data that is already in remote memory. For example, we typically see applications modifying only 1-8 cache-lines in a 4KB page, but the entire page is marked dirty and transferred over the network.

Overall, there is a mismatch between applications requirements for remote memory—low latency, fine-grain access, transparency—and the mechanism used to implement these requirements because virtual memory has not been designed to provide low-latency or fine-grain access. We need a different, more efficient mechanism to realize practical memory disaggregation.

2. Key Insights and Contributions

We identify two hardware primitives that are essential for memory disaggregation: 1) observing which data is accessed by the application, without incurring page faults; and 2) enabling fast dirty data tracking at cache-line granularity, without write page faults. *Our key observation is that both of these operations are trivially realized by the cache coherence protocol.* We describe an architecture using cache coherent FPGAs that leverages this observation to augment virtual memory in a new design for disaggregated memory. We designed and implemented Kona, a new software runtime that emulates these two new primitives using application instrumentation. In addition, we built two simulators that allow us to evaluate how much Kona can improve each remote memory operation.

We make the following contributions:

- We analyze the shortcomings of current software runtimes for remote memory and show that they result in large overhead and dirty data amplification.
- We propose a new approach for disaggregated memory software, using hardware primitives for remote memory caching and cache-line dirty data tracking based on cache coherence; we describe a reference architecture for these hardware primitives.
- We design and implement Kona, a software runtime that makes use of the new hardware primitives.
- We design and implement multiple emulation and simulation tools and we use them to measure the three different types of remote memory operations in Kona.

3. Key Results

Kona improves the execution time compared to virtual memory based systems by 6.6X (1 thread) and by 4-5X (2-4 threads) on a benchmark accessing random memory. Kona is faster because it improves all three core remote memory operations: 1) Kona improves remote memory caching by eliminating page faults, resulting in a reduced average memory access time by 1.7X and 5X compared to LegoOS and Infiniswap, respectively. 2) Kona improves the application performance during dirty data tracking by up to 35% compared to prior systems that

perform page granularity write-protection, while reducing the amplification by 2-10X using cache-line granularity. 3) Kona improves eviction efficiency by only writing modified cache-lines back to the remote memory, resulting in 4-5X better network goodput.

4. Main Artifacts

We designed and implemented Kona, a software runtime for disaggregated memory. Kona consists of a runtime library implemented in C, and two daemon processes: a rack resource manager for remote memory and a daemon that manages remote memory on every server. Other artifacts include an analysis of real applications' memory access patterns at cache-line granularity, realized using dynamic binary instrumentation. Kona relies on hardware primitives that are not yet available. Thus, we implemented two simulators that allow us to evaluate these primitives.

1) Fetching remote data. We developed KCacheSim, a cache simulator to measure the average memory access time for applications using remote memory.

2) Dirty data tracking. We developed KTracker to emulate cache-line granularity dirty tracking. KTracker attaches to a running process using *ptrace* and creates memory snapshots. Later, it compares the snapshots to find dirty cache lines.

5. Limitations of the State of the Art

A recent resurgence in software runtimes for remote memory has been fueled by low latency networks, the availability of Remote Direct Memory Access (RDMA) and new interconnects. These systems rely on OS-level mechanisms and interfaces, such as swapping or file systems, to offer remote memory to applications transparently [7, 2, 12, 9], without any applications modifications. However, the ease of programmability comes at the cost of forcing coarse-grain page-granularity access to remote memory through expensive OS code paths, leading to performance degradation and memory overhead.

A software system that provides access to remote memory has to support three main operations (§1). All three operations suffer from high overhead, due to their reliance on virtual memory. Next, we describe the overheads in more detail.

High overhead in fetching remote data. Remote memory systems incur large overheads due to page faults and TLB invalidations, causing large performance degradation. For example, we observed that moving as little as 25% of an application's data (Redis) remotely causes the throughput to drop by more than 60%. This degradation is caused by the high remote data access latency that all these systems incur: for example, LegoOS incurs a $10\mu s$ latency for a remote access, while Infiniswap incurs 40 μs. This high latency is astonishing, considering that a 4KB RDMA read operation is generally as fast as $3\mu s$, and comes mainly from the software stack.

Overhead in dirty data tracking and eviction. We measured a 35% decrease in throughput for Redis due to write page faults. The overheads are even higher for large pages, which first get broken down to 4KB pages to decrease the amplification [13]. Evicting pages adds its own overhead, since it incurs additional TLB invalidations on top of the ones required for dirty data tracking. We measured that eviction latencies could be over 32 μs with Infiniswap even though a 4KB RDMA write takes $3\mu s$.

High dirty data amplification. Often, applications only write to a small part of a page [1, 5]. However, remote memory systems track dirty data at page granularity, marking entire pages as dirty when only a small part is modified. This results in high amplification and poor network utilization, because more data is transferred over the network than necessary. We measured dirty data amplification for 4KB pages, 2MB pages, as well as for cache-lines (64 bytes) and determined that applications suffer from high dirty data amplification, as high as 31X for 4KB pages and 5500X for 2MB pages, respectively. In contrast, cache-line granularity tracking (64 bytes) results in a very small amplification (close to 1), suggesting that dirty data tracking is better done at cache-line granularity.

6. Why ASPLOS

Our work lies at the intersection between operating systems and computer architecture. State-of-the-art remote memory systems rely on virtual memory to offer remote memory to applications transparently. Our work analyzes the limitations of virtual memory for exposing remote memory and concludes that new hardware support is needed to push the boundaries of achievable performance. To this end, we leverage the cache-coherence protocol, which is one of the most fundamental concepts in computer architecture. Our proposed system continues to use virtual memory for translation and protection, as well as for failure handling. Thus, our design requires a careful interweaving of concepts from the two areas. In addition, our evaluation uses tools from multiple areas: zero-copy process communication and *ptrace*, dynamic binary instrumentation, and a cache simulator.

7. Citation for Most Influential Paper Award

This paper proposed a fundamentally new approach for disaggregated memory, based on tapping into existing information in current hardware – the cache coherence protocol. This approach was the first step in a long journey to make hardware disaggregated memory practical and efficient, by enabling a large part of the application data to reside remotely with only a small increase in memory access time. More broadly, the paper generated follow-on work on coherence-based runtimes for disaggregated memory that combined productive collaborations between researchers in operating systems, distributed systems and hardware architecture and eventually led to the adoption of disaggregated memory in every datacenter.

References

[1] Atul Adya, Robert Grandl, Daniel Myers, and Henry Qin. Fast key-value stores: An idea whose time has come and gone. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 113–119, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: a simple abstraction for remote memory. In *USENIX Annual Technical Conference (ATC)*, Boston, MA, 2018.

[3] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote memory in the age of fast networks. In *ACM Symposium on Cloud Computing (SoCC)*, 2017.

[4] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can far memory improve job throughput? In *European Conference on Computer Systems (EuroSys)*, 2020.

[5] Irina Calciu, Ivan Puddu, Aasheesh Kolli, Andreas Nowatzyk, Jayneel Gandhi, Onur Mutlu, and Pratap Subrahmanyam. Project PBerry: FPGA Acceleration for Remote Memory. In *Workshop on Hot Topics in Operating Systems (HotOS)*, page 127–135, 2019.

[6] Gen-Z draft core specification—december 2016. `http://genzconsortium.org/draft-core-specification-december-2016`.

[7] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient Memory Disaggregation with Infiniswap. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[8] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over commodity ethernet at scale. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, August 2016.

[9] Stefanos Kaxiras, David Klaftenegger, Magnus Norgren, Alberto Ros, and Konstantinos Sagonas. Turning centralized coherence and distributed critical-section execution on their head: A new approach for scalable distributed shared memory. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, page 3–14, New York, NY, USA, 2015. Association for Computing Machinery.

[10] Hasan Al Maruf and Mosharaf Chowdhury. Effectively Prefetching Remote Memory with Leap. In *USENIX Annual Technical Conference (ATC)*, 2020.

[11] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *Symposium on Operating Systems Design and Implementation (OSDI)*, Carlsbad, CA, 2018.

[12] Yizhou Shan, Shin-Yeh Tsai, and Yiying Zhang. Distributed shared persistent memory. In *ACM Symposium on Cloud Computing (SoCC)*, 2017.

[13] Mario Smarduch. Enhanced Live Migration For Intensive Memory Loads. `https://events.static.linuxfound.org/sites/events/files/slides/CloudOpen-Japan-2015.pdf`.

[14] Shin-Yeh Tsai and Yiying Zhang. LITE kernel RDMA support for datacenter applications. In *ACM Symposium on Operating Systems Principles (SOSP)*, October 2017.

[15] Erfan Zamanian, Carsten Binnig, Tim Harris, and Tim Kraska. The end of a myth: Distributed transactions can scale. 10(6), February 2017.