HoPP: Hardware-Software Co-Designed Page Prefetching for Disaggregated Memory

1. Introduction

Datacenter in-memory applications such as database and caching have an increasing demand to access large amounts of memory [1, 5, 6, 20]. Their performances degrade when their working set fail to fit into local memory. Unfortunately, servers are facing memory capacity walls due to pin and power limitations [**?**, 15]. Meanwhile the average memory utilization in datacenters is low (*e.g.*, about 60% for Google and Alibaba clusters [13, 21]), abundant idle memory is beyond the reach of applications that desperately need it.

Memory disaggregation can bridge this gap by organizing memory as an independent resource pool and making it available to applications. Disaggregation mitigates memory provisioning inefficiencies and improves resource utilization in datacenters [4, 7, 9, 11, 12, 18].

Generally, disaggregated memory systems have three key design goals. (1) *Transparency*. For ease of use, applications should be able to use the system with no or limited changes. (2) *Performance*. The system should deliver good performance even in the presence of networked memory accesses. (3) *Generality*. To enable a wider adoption, the system should support a broad range of datacenter applications written in distinct languages.

Unfortunately, no existing disaggregated memory systems can meet all design goals at once. In this paper, we take the following stand that ***it is possible to design a disaggregated memory system that is both transparent and general to run unmodified applications, all without performance compromises***. Our main approach is to solve the virtual memory bottleneck in a kernel-based system with a fundamentally different prefetching mechanism.

1.1. Limitations of the State of the Art

We first discuss the design spectrum for disaggregated memory systems (Table 1). We then present the limitations of existing prefetching solutions.

Disaggregated Memory System Design Spectrum. We categorize them into four types. (1) The *kernel-based systems* use virtual memory for transparent access to disaggregated memory [3, 14, 18]. But the costly kernel path takes a huge toll on performance. (2) In contrast, *application-integrated systems* sacrifice transparency for better performance via explicit user control on data movement and prefetching [16]. (3) *Language-runtime managed systems* seek a sweet spot in the middle by integrating disaggregated memory into a user space language runtime or application kernel. But they are limited to a few languages [22]. (4) *Bus-extended systems* rely on emerging coherent interconnects such as CXL for transparent remote accesses [3, 10]. However, they cannot use local memory for caching, which is critical for performance. Besides, most of them are still in their infancy.

Systems		Trans.	Generality	Performance		
				DRAM Cache	Light Data Path	Efficient Prefetch
App-Integrated [16, 17]		×	×	✓	✓	✓
Language-Runtime [22]		✓	×	✓	×	✓
Bus-Extended [10]		✓	✓	×	✓	✓
Kernel	*Others [2, 14]*	✓	✓	✓	×	×
	HoPP	✓	✓	✓	✓	✓

Table 1: *Disaggregated Memory System Comparisons.*

Existing Prefetching Limitations. Prior work has attempted to improve the kernel-based disaggregated memory system using prefetching [2, 14]. However, existing prefetching has the following limitations. First, they train the algorithms using address from the page fault. However, this address is coarse-grained and filled with interference noise. Moreover, page faults may occur infrequently results in less data for prediction. Second, existing prefetching subsystem has inefficiencies. In particular, prefetched pages will still trigger page faults because the page table is not established during prefetching.

1.2. Key Insights

We observe that the fundamental cause for existing prefetching is due to the semantic gap between an operating system (OS) and its applications: OS cannot know where (which memory address) an application is running at until page faults.

This motivates us to decouple page faults from prefetching by capturing the memory access trace at the memory bus and feeding it to the OS continuously. As a result, a prefetch algorithm has abundant supply of real-time addresses to make better predictions, independent from how frequent page faults occur. In other words, prefetching can reduce the number of page faults without worrying its prediction quality.

More importantly, this finally enables us to close the performance gap for kernel-based disaggregated systems using prefetching. In specific, we can separate prefetching from the traditional page fault, running as an independent data path. Moreover, we can design advanced prefetch algorithms with high accuracy/coverage using the full memory access trace. Finally, we can maximize the bene-

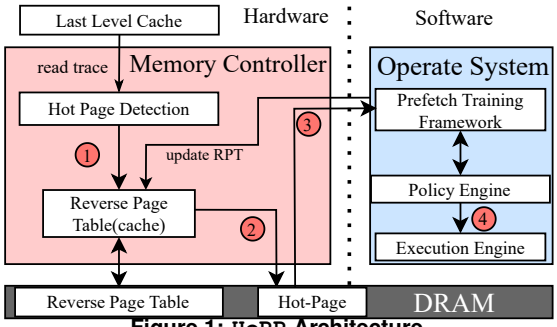

Figure 1: HoPP Architecture.

fit of prefetching by pre-populating the page tables.

2. Design and Implementation of HoPP

We designed and implemented HoPP, a **H**ardware-software c**o**-designed **P**age **P**refetching system. HoPP is an in-kernel RDMA-based disaggregated memory system that meets all design goals. HoPP relies on prefetching to reduce the number of page faults thereby improving performance. It has two main parts. We implement a hot page detection and a reverse page table cache in hardware. By design, both hardware modules are integrated into a memory controller. The software part consists of a training module, a policy engine and an execution engine. Figure 1 presents overall architecture .

We now describe HoPP in details.

- HoPP's input is raw memory accesses from the last-level cache (LLC) misses. We use the *hot page detection (HPD)* module to reduce bandwidth usage, filter cold pages, and find the hottest ones. Specifically, it translates cache line-sized LLC miss into page-level trace. It uses a small cache to track the most accessed pages in a configurable time window. Once a page's access frequency exceeds a threshold, it emits the physical page number (step 1).
- We propose *reverse page table (RPT)* to map the emitted physical page number back to a specific application (PID) and its virtual address (VA). The identified information is further saved into a known location in local memory (step 2). Though designed in a compact format, the whole RPT cannot fit into hardware. Thus, we save it to local memory and add a small caching module in hardware to accelerate the mapping.
- We build a stream-based prefetching algorithm using the hot pages from the RPT (step 3). It identifies access patterns and accurately predicts what pages to prefetch. We use a *stream training table* to group PID and VA records into page streams, which are then used to detect representative patterns such as stride [14].
- The *prefetch policy engine* has two knobs to tweak prefetch aggressiveness, thereby minimizing prefetching's impact on foreground performance. We use *prefetch intensity* to determine how many pages to prefetch. We use *prefetch offset* to control how far in time should we prefetch.
- Finally, we build a *prefetch execution engine* to cleanly separate prefetching from the page fault, running as an independent data path. The execution engine is responsible for reading data from remote over RDMA. It populates the page table once the data returns.

We prototype HoPP on top of commodity servers. Since the memory controller is vendor-locked, we deploy a hardware-based memory tracking tool called HMTT [8] as a bump-in-the-wire between the memory controller and the DRAM chips. It can capture and output the full memory access trace. Consequently, we emulate the HPD and RPT modules in software. We also implement them in Verilog for hardware area and power estimation. HoPP's networking infrastructure is adopted from Fastswap [2].

3. Results and Contributions

We evaluate HoPP with 15 real-world large in-memory applications such as GraphX and K-means. Our evaluation indicates HoPP can predict prefetching with high accuracy and coverage. When half of their working set is disaggregated, applications running on top of HoPP only incur 3.53% slowdown with 99.5% coverage and 99.9% accuracy, a 59% improvement over Fastswap and Leap.

We further use CACTI [19] to estimate the area and static energy expenses for our proposed modules. All of them are power-efficient and occupy trivial chip area.

In summary, we make the following contributions:

- We show that with the help of an improved prefetching, it is possible to design a disaggregated memory system that offers transparency and generality, all without performance compromises.
- We build HoPP. To the best of our knowledge, HoPP is the first disaggregated memory system that meets all design goals
- We propose to decouple page fault from prefetching and demonstrate its feasibility using a special hardware tracking tool.
- We propose two practical and efficient hardware modules, hot page detection and reverse page table, to capture real-time page access information.
- We propose a stream-based prefetching algorithm to take advantage of the full memory trace and two metrics to control prefetch aggressiveness.
- We demonstrate HoPP using real hardware and show extensive evaluation using large in-memory applications.

References

[1] Memcached - a distributed memory object caching system. http://memcached.org.

[2] 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 *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, 2020.

[3] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of*

the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2021, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, 2016. USENIX Association.

[5] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, Hollywood, CA, 2012. USENIX Association.

[6] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 599–613, Broomfield, CO, 2014. USENIX Association.

[7] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA, 2017. USENIX Association.

[8] Yongbing Huang, Licheng Chen, Zehan Cui, Yuan Ruan, Yungang Bao, Mingyu Chen, and Ninghui Sun. Hmtt: A hybrid hardware/software tracing system for bridging the dram access trace's semantic gap. *ACM Trans. Archit. Code Optim.*, 11(1), 2014.

[9] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dredbox project vision. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 690–695, 2016.

[10] Huaicheng Li, Daniel S Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Ishwar Agarwal, Mark Hill, Marcus Fontoura, et al. First-generation memory disaggregation for cloud platforms. *arXiv preprint arXiv:2203.00241*, 2022.

[11] Shuang Liang, Ranjit Noronha, and Dhabaleswar K. Panda. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *2005 IEEE International Conference on Cluster Computing*, 2005.

[12] 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 *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, Austin, Texas, 2009.

[13] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the cloud: An analysis on alibaba cluster trace. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2884–2892, 2017.

[14] Hasan Al Maruf and Mosharaf Chowdhury. Effectively prefetching remote memory with leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 2020.

[15] Matthew Poremba, Itir Akgun, Jieming Yin, Onur Kayiran, Yuan Xie, and Gabriel H. Loh. There and back again: Optimizing the interconnect in networks of memory cubes. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, 2017.

[16] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. AIFM: High-performance, application-integrated far memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020.

[17] Daniel J. Scales, Kourosh Gharachorloo, and Chandramohan A. Thekkath. Shasta: A low overhead, software-only approach for supporting fine-grain shared memory. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, Cambridge, Massachusetts, USA, 1996.

[18] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, 2018. USENIX Association.

[19] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. *wrl research report*, 2001.

[20] M. Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 2013.

[21] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A memory-disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020.