Challenges in Building and Deploying Disaggregated Persistent Memory

Yizhou Shan, Yutong Huang, Yiying Zhang
Purdue University
{*ys, huang637, yiying*}*@purdue.edu*

Byte-addressable non-volatile memory (NVM) such as 3D-Xpoint and the memristor provides persistence, close-to-DRAM performance, and high density. Apart from packaging NVMs in SSDs and using them as storage devices, NVMs can be used as memory (e.g., attached to and accessed directly from the memory bus). These usage models are often called non-volatile main memory or *persistent memory (PM)*. In 2018, Intel released Optane PM and it is recently made available in the Google Cloud [2].

The first and a fundamental issue to be solved before PM can be used in datacenters is how they should be deployed. Existing proposals have all used the model of attaching PMs to the main memory bus of commodity servers. However, such model requires datacenters to purchase and install new servers that can host PMs, since datacenter vendors usually pushes for full hardware utilization of servers and do not leave empty DIMM slots in their existing servers.

We believe that a more cost-efficient and flexible way to deploy PM is to build PMs as stand-alone devices and to attach them directly to the datacenter network, a model we call *disaggregated persistent memory*, or *DPM*. Each DPM device has a network interface, a PM controller, and bulk PM. DPM shares many benefits with a more general disaggregated datacenter architecture [3]. For example, DPM requires no general-purpose cores and all its functionalities can be implemented in one hardware board. Another submission from our group [4] describes the benefits, challenges, and software solutions of DPM.

This work addresses the hardware and networking challenges in building and deploying DPMs. Below, we discuss the challenges we foresee and potential solutions we propose.

Network interface of DPM. Host machines access DPMs directly through the network; thus each DPM need to implement a network interface. Modern datacenter NICs and OS network stacks support many network functionalities that are heavy-weight and not necessary for DPMs. Host machines should be able to access remote DPMs with low latency and high bandwidth to match PM's DRAM-like performance.

A unique feature of DPM that we can leverage in our network interface design is that DPMs are only the *passive* receiver of requests from host machines and do not initiate new network requests or forward network requests. This property implies that DPM does not need to perform flow control or packet forwarding. Moreover, DPMs (more specifically, the PM controlling part of DPMs) directly consume incoming network requests and there is no need to support multiple VMs or processes as consumers or complex queue/load management.

We believe that we should build new lightweight, customized network interfaces for DPM. Similar to the Catapult's Lightweight Transport Layer design [1], we only need to implement the most basic network functionalities in DPM, including basic connection management, header management, and reliable network delivery. To improve DPMs' throughput and tail latency, we believe that DPMs should employ a large number of parallel units to handle network requests, but each of them will be lightweight and low-power.

Interface and API implementation. DPMs should support a minimal yet flexible set of APIs for host machines to build various PM-based software: *e.g.*, making connections, space allocation, read and write. A new challenge is the protocol to ensure data persistence: data not only needs to be received by the network interface, but also written all the way to the NVM media. DPM should support new operations for making NVM writes persistent (*i.e.*, data is written all the way to the physical NVM media instead of in volatile caches or buffers). Such operations can either be a *persistent write* operation where each write is guaranteed to be persistent or a *persistent flush* operation which flushes all outstanding writes to be persistent.

Addressing. There are three options of addressing PM in DPMs. First, host machine applications can use *virtual memory addresses* and a host-side library or kernel maintains the mapping from these virtual addresses to physical memory addresses of DPMs. Hosts send physical addresses to DPMs, which then directly access PM using these addresses. A similar approach is to instead maintain the mapping at DPMs and let host machines send virtual addresses over the network. The final option is to directly expose physical addresses to host applications.

There are different trade-offs for each of these options. The first and the third options remove the need for DPMs to maintain any address mappings, making DPMs simpler and much cheaper to build. However, every time when a DPM moves data around (*e.g.*, due to load balancing or wear leveling), host machines need to be informed. The second option requires DPM to maintain mappings (either in a DRAM attached to the main DPM hardware

board or in on-board SRAM), which will increase DPM's monetary and energy costs. The third option needs no mapping at all and is the cheapest method. However, we will need to employ new protection mechanisms that can work with physical addresses directly.

Apart from the location of maintaining address mapping, another open issue is the granularity of address mapping. All common modern architectures use paging to maintain virtual memory (usually in 4KB granularity and optionally in huge 2MB or 1GB pages). With PM's high density, we anticipate DPMs to have TB-level capacity (Google already provides virtual machines with 7TB PM [2]). Paging will cause either high mapping space overhead (small page size) or internal fragmentation (huge page size). Since we are building DPM architecture from scratch, we will have the freedom to choose memory mapping granularity and mechanism, for example, with segmentation.

Space allocation and operation scheduling. Applications need to allocate and de-allocate PM space in DPMs. Instead of letting host machines manage PM space, it is easier and more efficient to have DPMs manage their own PM space. To fit the limited hardware resource of DPM devices, DPMs should use simple allocation mechanisms that require small metadata.

We anticipate PMs to have large internal parallelism. DPMs need to manage and schedule operations across these internal parallel units. There are three goals of DPM operation scheduling. First, DPMs should schedule requests to deliver good performance, for example, by scheduling requests to as many parallel PM units as possible to improve throughput. Second, DPMs should guarantee tail latency for applications that require SLOs. Finally, since many PM-based software systems require or desire ordering of I/O operations, DPMs should perform their scheduling in a way that still preserve user-requested ordering points.

Hardware platform. There are several hardware options to implement a DPM device, including ASIC, FPGA, and SoC. Each of these hardware platform has its pros and cons. ASICs offer high performance and low power consumption. However, the development cycle of ASIC-based solutions is costly and lengthy. Moreover, ASICs only provide fixed function after manufacturing. DPMs can benefit from ASICs' performance advantages, but ASICs come short when users of DPM want to implement new features in DPMs.

SoCs contain one or more general-purpose, low-power cores and can run more complex software and full-fledged OSes. Thus, SoC-based DPM solutions enjoy the same flexibility benefits as software-based solutions. However, SoCs are generally slower than their hardware counterparts.

FPGAs are reconfigurable hardware that meet the flexibility needs of many modern datacenter applications and they have been deployed in several major clouds [1]. FPGAs are more power-hungry and less performant than ASIC, but FPGAs can efficiently accelerate many logics that have inherent parallelism. FPGA solutions are also much faster to develop than ASICs. An FPGA-based DPM has software-like flexibility and hardware-like performance and could be a viable solution for future DPMs in datacenter environments.

Large-scale deployment and network topology. A final challenge to be solved is the actual deployment of DPMs in existing datacenters. Being able to directly attach to the network makes it easier to connect DPMs, but we still need to change existing network topology or potentially build a new one. Where DPMs are placed in a topology and how far (in network hops) they are from host machines will largely affect application performance.

The easiest way is to attach DPMs to existing ToR switches. But existing ToR switches may not have enough free ports or the locations of free ports are far from other host machines. Designing new topologies for DPMs allow customization and potentially better performance or cheaper monetary cost, but doing so will require changes to existing datacenter topologies.

There are more challenges in deploying DPMs in large-scale datacenters. For example, a common use model is to have multiple host machines share a DPM device, which can create network congestions. With the scale of one thousand nodes or beyond in a datacenter, network delay can largely impact application performance when host machines have to cross a few switches to access DPMs.

Conclusion and future work. This document presented a set of challenges we foresee in building and deploying DPMs and solutions that we proposed. We believe that it could be useful for future researchers and practitioners who work on PMs in datacenter environments. We are currently in the middle of implementing our own FPGA-based DPM solutions and we hope to be able to present a more concrete solution at NVMW'19.

References

[1] Adrian M. Caulfield et. al. A cloud-scale acceleration architecture. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '16)*.

[2] Google. Available first on Google Cloud: Intel Optane DC Persistent Memory. goo.gl/uNjWG2.

[3] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*.

[4] S.-Y. Tsai and Y. Zhang. Building Atomic, Crash-Consistent Data Stores with Disaggregated Persistent Memory. In *NVMW'19 Submission 41*.