Topics in Persistent Memory

> Author: Guanzhou (Jose) Hu 胡冠洲 @ UW-Madison CS839
>
> Teacher: Prof. Michael Swift

This note includes the summary paragraphs I wrote in reviews for each paper. For detailed information about their contributions, strengths, and weaknesses, please refer to the slides and our HotCRP site (private).

The following terminologies are used interchangeably, though they might mean slightly different things in specific papers:

- *Non-volatile memory* (NVM)
- *Persistent memory* (PM, PMEM)
- *Storage-class memory* (SCM)

NVM Hardware

Measurement: Optane DIMMs

Link: https://www.usenix.org/conference/fast20/presentation/yang

This paper presents an empirical performance study of *Intel Optane DIMM* devices. In particular, it focuses on the use of Optane DIMMs as persistent memory in *AppDirect mode* and compares it with the well-known performance characteristics of traditional volatile DRAMs. Results suggest that Optane DIMMs have different performance behaviors than DRAMs in terms of latency, tail latency, and R/W bandwidth on different access sizes and concurrency. Based on the results, the paper summarizes 4 empirical rules for using Optane DIMMs and provides case studies on current PM systems to demonstrate the rules.

Measurement: Optane SSDs

Link: https://research.cs.wisc.edu/adsl/Publications/hotstorage-contract19.pdf

(Auxiliary reading) The paper presents an empirical performance study of *Intel Optane SSD* devices.

Wear Leveling: Start-Gap

Link: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.7213&rep=rep1&type=pdf

The paper proposes a new phase-change memory (PCM) wear-leveling technique called *Start-Gap*, which achieves 97% of normalized endurance meanwhile having low computation and storage overhead. Start-Gap deploys a simple algebraic calculation of `PA = (LA + start) % N (+ 1 if above gap)`, which requires only two extra hardware registers and one extra gap line. Gap is shifted every tunable ψ writes to enable uniform weal-leveling. The paper further extends Start-Gap with address randomization to tolerate hot-spot locality, and with region partitioning to defend against thrashing attacks.

PCM as Main Memory Extension

Link: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.6330&rep=rep1&type=pdf

The paper proposes a hybrid main memory system organization, where a small DRAM buffer is placed in front of denser but slower *phase-change memory* (PCM). By applying lazy-write DRAM caching, line-granularity writebacks, fine-grained wear-leveling, and page-level bypass techniques, the paper shows that such hybrid memory architecture is able to achieve comparable performance with pure DRAM of the same size, meanwhile being 4x denser in capacity, reducing page faults by up to 5x, and consuming nearly 50% less power.

PCM as SSD Storage Array

Link: https://ieeexplore.ieee.org/document/5695552

The paper presents *Moneta*, a storage array design for PCM devices. Moneta comprises of a central scheduler implementing PCIe interface and DMA buffers, a ring of 4 FPGA memory controllers, and several PCM banks attached to each memory controller. Beyond the baseline design, the paper introduces a sequence of software/hardware optimizations that greatly improve Moneta's latency and throughput performance, including bypassing the Linux IO scheduler, enabling one 64-bit atomic PIO per request, doing spin polling for completion, using separate DMA read/write queues, and doing block-based round-robin scheduling. Evaluation results show that Moneta is able to almost saturate a 2GB/s full-duplex PCIe link, and outperforms existing storage array solutions based on HDDs or SSDs.

PM File Systems

Kernel: BPFS

Link: https://www.sigops.org/s/conferences/sosp/2009/papers/condit-sosp09.pdf

The paper presents BPFS, a file system over persistent memory DIMMs. BPFS achieves good latency performance and guarantees crash consistency through *short-circuit shadow paging*, a PM-optimized shadow paging technique. It suggests that hardware should provide two features: 64-bit atomic writes and epoch barrier instructions. With these features, shadow paging can be localized to a subtree where the change at the top is within 64 bits. Copy-on-write can be avoided in special cases of in-place updates and in-place appends.

Kernel: PMFS

Link: https://courses.engr.illinois.edu/ece598ms/fa2019/papers/paper45.pdf

This paper presents the design and implementation of PMFS, a practical file system for PM DIMMs aside volatile DRAM. For crash consistency, PMFS proposes a *pm_wbarrier* instruction for ensuring durability through memory controller, and deploys a *hybrid consistency approach* combining in-place updates, journaling, and copy-on-write. PMFS maps the entire PM into kernel memory address space and introduces *write protection windows* to prevent kernel bugs from corrupting PM data. To optimize for mmap, PMFS actively uses large pages to reduce paging overhead. The paper also describes a PMFS validation tool named Yat. Evaluation results show that these design choices collectively bring significant performance gain to PMFS as compared to Ext2/4 on PM block drivers.

Kernel: NOVA

Link: https://www.usenix.org/conference/fast16/technical-sessions/presentation/xu

This paper presents NOVA, an in-kernel log-structured FS optimized for PM DIMMs. NOVA maintains a linked-list inode log for inode metadata, where updated entries are appended to the log and committed atomically. A garbage collector periodically cleans and compacts the log. NOVA does copy-on-write for data, where stale data pages are reclaimed immediately to further reduce log size and simplify garbage collection. Evaluation results show that NOVA performs other file systems on PM under most workloads.

Kernel: NOVA-Fortis

Link: https://cseweb.ucsd.edu/~swanson/papers/SOSP2017-NOVAFortis.pdf

(Auxiliary reading) Fault-tolerance enhancement to NOVA.

Kernel: Ext4-DAX

Link: https://www.kernel.org/doc/Documentation/filesystems/dax.txt

(Auxiliary reading) *Direct-access* (DAX) technique in traditional Linux filesystems.

Userspace: Aerie

Link: https://dl.acm.org/doi/abs/10.1145/2592798.2592810

(Auxiliary reading) Managing both file data and most of file metadata in userspace, through a trusted FS service daemon process.

Userspace: ZoFS

Link: https://ipads.se.sjtu.edu.cn/_media/publications/dongsosp19-rev.pdf

(Auxiliary reading) Granting userspace library filesystem direct control of file metadata through the new abstraction of *Coffers*.

Hybrid: Strata

Link: https://www.cs.utexas.edu/users/witchel/pubs/kwon17sosp-strata.pdf

(Auxiliary reading) Using a per-process private log space to buffer updates, trading off visibility for performance. Migrating hot-cold data across different layers of storage devices.

Hybrid: SplitFS

Link: https://dl.acm.org/doi/pdf/10.1145/3341301.3359631

The paper presents SplitFS, a PM file system that uses in-kernel Ext4-DAX for all *metadata* operations and bypasses the kernel through a memory-mapping user library for all *data* operations. To handle the case of appending a file, SplitFS steers new data to a process-private staging file and introduces a *relink* syscall, triggered on `fsync()`, that links the new blocks to file metadata without copying. Evaluation results show that this split architecture brings ~2x better performance than both in-kernel PM-optimized FS such as original Ext4-DAX and NOVA, and user-level kernel-bypassing FS such as Strata.

Virtual Memory Optimizations

VM For PM: pVM

Link: https://dl.acm.org/doi/10.1145/2901318.2901325

This paper presents pVM, an extension to the OS virtual memory system to support PM. pVM abstracts PM as a NUMA memory node instead of a block device behind VFS. This design allows pVM to support flexible and efficient memory scaling when DRAM capacity is under pressure. It also outperforms VFS-based systems in the use case of a persistent object store due to better cacheline- and TLB-efficiency. Evaluation results show that pVM brings up to 2.5x speedup to memory-intensive applications and results in significantly fewer cache and TLB misses of OS data structures.

Better `mmap`: DaxVM

Link: not available yet

This paper presents DaxVM, a set of optimizations to the OS memory mapping interface that reduces the software overhead of mapping/unmapping for PM DAX use cases. DaxVM maintains pre-populated file page tables that enable O(1) `mmap()`, supports ephemeral mappings, aggregates `munmap()`'s asynchronously, and drops kernel `msync()` support. Evaluation results show that these optimizations/reductions make memory mapping a solid choice for applications that frequently access multiple small files for short durations, where `read()`'s were favored.

Better `mmap`: HashFS

Link: https://web.eecs.umich.edu/~takh/papers/neal-fast-2021.pdf

(Auxiliary reading) Hash-based `mmap` to provide memory mapping lower latency.

PM Supporting Frameworks

Software: Mnemosyne

Link: https://pages.cs.wisc.edu/~swift/papers/asplos11_mnemosyne.pdf

This paper presents Mnemosyne, a complete programming framework for user-space access to PM. Mnemosyne enables user-mode direct access to PM by discarding the FS abstraction. It ensures the correctness and consistency of modifying persistent data through virtualization and a rich stack of programming interfaces, including HW primitives, persistent regions, persistent heap, logging, and transactions. Evaluation results show that Mnemosyne outperforms Berkeley DB on ramdisk.

Software: Atlas

Link: https://www.hpl.hp.com/techreports/2013/HPL-2013-78.pdf

This paper presents Atlas, a programming framework for converting multi-thread lock-based programs to durable programs on PM. Atlas exploits the fact that critical sections in lock-based programs already encode the boundary of transactions to ensure PM consistency, and introduces the notion of *failure-atomicity sections* (FASEs) that can be inferred from lock-protected critical sections. At compile time, Atlas injects calls to library functions to do undo logging of all PM stores per thread. At runtime, a helper thread does garbage collection of log entries. When recovering after a crash, Atlas parses the log entries to identify the latest consistent state and undoes later updates. Atlas is able to automatically convert correct lock-based concurrent programs to durable PM programs, but in the cost of potentially large overhead.

Software: PMDK

Link: https://pmem.io/pmdk/

(Auxiliary reading) The most popular PM programming library developed and maintained officially by Intel.

Hardware: Mojim

Link: https://cseweb.ucsd.edu//~yiyingzhang/mojim-asplos15.pdf

The paper proposes Mojim, an NVM driver-layer service that uses primary-backup replication to improve the reliability of NVM storage in data centers. Mojim deploys a two-tier architecture, where the top tier consists of a primary node and a mirror node and the optional secondary tier contains backup nodes. Mojim provides upper-layer file systems with a different `msync` semantic for NVM: instead of flushing CPU cachelines, it replicates the synced area to the mirror node through fast RDMA. The mirror node lazily propagates updates to backup nodes in the background. Evaluation results show that Mojim provides lower latency than original `msync` by avoiding cache flushes on the critical path, and has comparable performance to unreplicated deployment.

Hardware: ThyNVM

Link: https://cseweb.ucsd.edu/~jzhao/files/thynvm-micro2015.pdf

This paper presents ThyNVM, a software-transparent solution to NVM consistency on hybrid DRAM+NVM architecture. ThyNVM requires no software modification (journaling or shadowing) by doing *checkpointing* at memory hardware level. To reduce the overhead of checkpointing, ThyNVM overlaps checkpointing of the last epoch with execution of the current epoch. ThyNVM makes a tradeoff between checkpointing speed and storage overhead by combining DRAM page-granularity writeback scheme and NVM cacheblock-granularity remapping scheme, and switching between the two schemes dynamically based on access locality. Evaluation results show that ThyNVM achieves comparable performance with software-integrated journaling or shadowing solutions.

Data Structures On PM

DS Designs: CDDS

Link: https://shivaram.org/publications/nvm-fast11.pdf

The paper proposes CDDS, an efficient versioning-based data structure design scheme for NVM. CDDS advocates for building single-level data structures directly on byte-addressable NVM devices. CDDS uses versioning, where any update to the data structure writes entries to new location with new version number and advances the global version number at commit. A garbage collector cleans older versions with no references. The paper describes in detail a CDDS B-tree implementation, which performs better than previous logging-based durable B-trees. Tembo, a Redis-based KV store with CDDS B-tree, yields significantly higher throughput than Cassandra.

Conversion: RECIPE

Link: https://www.cs.utexas.edu/~jaya/pdf/sosp19-recipe.pdf

The paper presents RECIPE, a principled approach for converting concurrent DRAM data structures to crash-consistent PM data structures. RECIPE exploits the strong similarity between concurrent data structures' *isolation* requirements and PM's *crash consistency* requirements. When certain conditions are met, a DRAM data structure can be easily ported to PM by modifications as simple as adding clflushes and fences after stores. The paper demonstrates five example data structures converted with RECIPE, verifies crash consistency of outcomes with a new testing method, and shows their performance excellence over state-of-the-art data structures.

PM Programming Safety

Statically-Enforced: Corundum

Link: https://cseweb.ucsd.edu/~mhoseinzadeh/hoseinzadeh-corundum-asplos21.pdf

This paper presents Corundum, a Rust library for PM programming that enforces many PM safety invariants statically at compile-time. Corundum is strictly based on transactions. It puts rules on references to persistent objects across pools and references to volatile/persistent objects across transaction boundaries. To enforces most of the rules statically, Corundum exploits Rust's strong type system and extends Rust's smart pointer types. Evaluation results show that Corundum can achieve comparable (and sometimes better) performance to existing PM programming frameworks such as PMDK and Atlas.

Bug Finding: PMTest

Link: https://akolli.github.io/pubs/pmtest-asplos19.pdf

This paper presents PMTest, a fast and flexible framework for testing durability and ordering in PM crash-consistent software. PMTest is flexible because it provides programmers with two generic checker interfaces: `isPersist()` and `isOrderedBefore()`, to assert guarantees that the program must meet. PMTest is fast because it avoids exhaustive enumeration and implements the two checkers by tracking all the store, flush, and fence operations, maintaining a global epoch number marked by fences, and inferring *persistent intervals* of objects within which they could be made persistent. The checking engine runs in pipeline with the foreground PM program to further improve performance. Evaluation results show that PMTest runs ~7.1x faster than pmemcheck and is able to find new bugs in PMFS and PMDK B-tree.

Bug Fixing: Hippocrates

Link: https://web.eecs.umich.edu/~barisk/public/hippocrates.pdf

This paper presents Hippocrates, an automated PM bug-fixing tool that is guaranteed to not introduce new correctness issues and is best-effort in performance. Hippocrates takes in the trace output of a bug-finding tool and identifies the location of missing flushes/fences. It first attempts to perform *intraprocedural* fixes, where all missing flushes/fences are inserted inline. To improve performance, it then tries to convert the inserted flushes/fences into *interprocedural* fixes by hoisting them up the function stack. The conversion is guided by a heuristic algorithm that minimizes the influence on volatile codepaths and is implemented through *subprogram transformation*. Hippocrates is able to automatically fix 23 bugs in various PMDK applications and deliver comparable performance with manual fixes.

Memory Persistency Theory

Memory Persistency

Link: https://web.eecs.umich.edu/~twenisch/papers/isca14.pdf

This paper presents the theoretical notion of *memory persistency*, an extension to *memory consistency* for NVM. In addition to `load`s and `store`s, memory persistency models involve `persist` operations, and specify constraints on how persists must be ordered with respect to stores and other persists. The paper describes three specific memory persistency models, *strong persistency*, *epoch persistency*, and *strand persistency*, in the order from the strongest to the most relaxed. Simulation results show that relaxed persistency ordering could potentially improve performance by a large margin for a queue data structure.

Delegated Persist Ordering

Link: https://aasheeshkolli.files.wordpress.com/2016/08/delegated-persist-ordering-micro16.pdf

This paper proposes delegated persist ordering, a PM ordering design that offloads ordering constraints to the PM controller to improve performance. Previous *synchronous ordering* (SO) designs require explicit `clwb`, `pcommit`, and `sfence` instructions that stall execution on the critical path. Delegated ordering removes the need of calling flush instructions and hides the latency of flushing, by tracking write dependencies in per-core data caches and communicating fence operations to the PM controller. Buffering thus happens at two levels: 1) writes from one thread could aggregate in the per-core *persist buffers*, with the order of writes respected, until a dependent fence is forced, and 2) flushed writes from all cores could further aggregate at the global *write buffer* in PM controller until a fence operation is received. Evaluation results show that delegated ordering improves latency by up to 3.73x over SO.

NVM Usage In Databases

Three-Tier Buffer Manager

Link: https://db.in.tum.de/~leis/papers/nvm.pdf

This paper presents a three-tier database buffer manager design that exploits the performance benefits of byte-addressable NVM and supports graceful degradation as data size grows larger. The three-tier architecture places DRAM over NVM to form a two-tier cache over SSDs. Between DRAM and NVM, data is mapped in *cacheline granularity* to enjoy the byte-addressability of NVM, and *mini pages* are used to reduce DRAM space usage. When data is evicted from DRAM, an *admission set* algorithm decides on NVM admission to try to keep warm (but not hot) data in NVM. Additional techniques, including *pointer swizzling*, *combined page table*, and *mapping table reconstruction* at restart, are deployed to further improve performance. Evaluation results show that the three-tier design outperforms both NVM-direct databases and page-granularity buffer managers, meanwhile allowing large datasets to be kept on SSDs.

SAP HANA Adoption

Link: http://www.vldb.org/pvldb/vol10/p1754-andrei.pdf

This paper summarizes the design decisions made to integrate NVRAM into the SAP HANA database. NVRAM is placed beside disk storage and is used to store only the Main Column Fragments data, which are sequential and are rarely updated, to avoid drastic changes to the codebase and to reduce the sync-up overhead between DRAM and NVRAM. The column data format adjusts memory alignment and avoids complex deserialization to improve cache performance. NVM blocks have an optimized lifecycle that discards logging and relies on the atomicity of pointer switching at commit (i.e., shadow copying). Experiment results show that HANA achieves better recovery performance with NVM than cold starts from DRAM+disk.