Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

Data management on non-volatile memory

by Joy Arulraj and Spyros Blanas on Sep 3, 2018 | Tags: Databases, Memory, non-volatile

In a recent blog post, Steve Swanson presented three milestones that mark the adoption path of non-volatile memory by applications. The next step in this path is tailoring fundamental protocols and algorithms for non-volatile memory. We illustrate this point using recent research results from the database systems community.

The advent of non-volatile memory (NVM) invalidates fundamental design decisions that are deeply embedded in today's software systems. In a recent blog post, Steve Swanson presented the three milestones that mark different stages of NVM adoption. The proposed steps would certainly enable general applications to leverage NVM

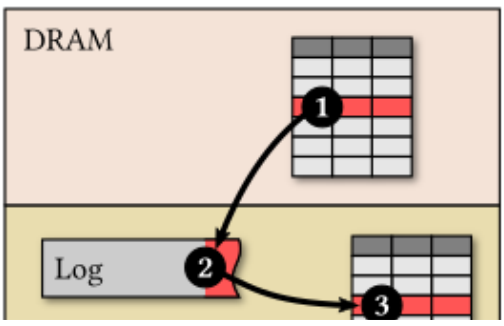
with limited development effort. However, recent research in the database systems community points out that the opportunity for NVM-aware applications does not stop with bespoke NVM-aware data structures. We believe that the next level of evolution, or level 4.0 in the original milestones, is tailoring fundamental protocols and algorithms for NVM.

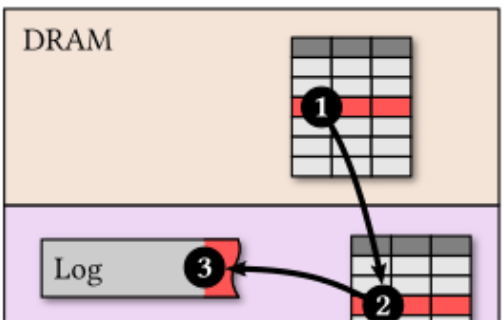
This blog post details how database systems are being redesigned for NVM. We first present a new logging and recovery protocol for NVM and then describe a data processing algorithm for sorting NVM-resident data. We hope that this blog post will give the computer architecture community a better idea of the different ways in which NVM may be used by data-intensive applications.

A Logging and Recovery Protocol for NVM

The write-ahead logging protocol is commonly used to ensure that the data stays consistent after a system crash. This protocol is designed with the assumptions that memory is fast but volatile and that durable storage only supports slow random writes. With write-ahead logging, one first records the updates made by a transaction in a log on durable storage using sequential writes. It later propagates the changes to the database on durable storage using random writes outside the critical path of transaction processing. While recovering from a system crash, the application replays the log to restore the database to a consistent state.

Merely replacing durable storage (e.g., SSD) with NVM and operating on a NVM-aware file system improves the runtime performance of the database system by 10X. This is due to the inherent performance advantage of NVM and the file system optimizations that exploit the byte-addressability property of NVM. However, the recovery cost and the storage footprint of the database system remain unchanged. The key insight is that NVM invalidates the fundamental design assumptions behind the write-ahead logging protocol, since random writes in NVM are both persistent and fast. With NVM, the write-ahead logging protocol suffers from two problems: (1) unnecessary duplication of data in the log and in the database, and (2) slow recovery from system crashes.





Write-ahead logging

Write-behind logging

Recent work tailors the logging and recovery protocol for NVM to address these problems. Since NVM supports fast random writes, the database system can directly flush changes to the database on NVM during regular transaction processing, instead of first recording the updates on the log. Unlike the write-ahead logging protocol, the database system persists changes made by transactions in the database **before** marking those transactions as having been committed in the log. When recovering from a system crash, the system needs to identify the changes made by the transactions running at the time of failure. To accomplish this, it suffices to only record metadata about the running transactions in the log. Due to this flipped order of writes between the log and the database, this protocol is referred to as write-behind logging.

Write-behind logging solves the two problems of write-ahead logging. First, it eliminates data duplication between the log and the database by only recording meta-data in the log. Second, it enables near-instantaneous recovery from system crashes. With write-behind logging, the system keeps track of the running transactions in the log. During recovery, it reads the latest log record to determine which transactions were running at the time of failure and then ignores the effects of these transactions while reading the database. It is not necessary to replay the entire log since all the modifications of committed transactions are already present in the database. This protocol enables the database system to recover 100X faster than write-ahead logging and cuts the storage footprint nearly in half.

Write-behind logging illustrates the importance of tailoring protocols for NVM. We next turn our attention to how data processing algorithms can be adapted for NVM.

Data Processing Algorithms for NVM

Common data processing algorithms, like sorting, are designed to have low computational complexity and to exploit the processor caches of modern multi-core CPUs. These algorithms are designed with the assumption that reads and writes to memory take the same amount of time to complete. However, NVM invalidates this assumption, as in certain NVM technologies, writes take longer to complete compared to reads. Further, excessive writes to a single memory cell can destroy it. Given the read-write asymmetry and limited write endurance of NVM, the data management community has recently considered how to design data processing

algorithms that limit the number of memory writes.

Let us consider the sorting algorithm in more depth. To efficiently order larger-than-memory datasets, database systems employ external sorting algorithms that make multiple passes over the data, such as the external merge sort algorithm. This algorithm starts by splitting the input data into chunks that fit in main memory, then uses an in-memory sorting algorithm to sort the tuples within the chunk, and then writes the sorted chunks back to durable storage. These sorted chunks are successively merged across multiple passes to produce the final sorted data in multiple merge passes. From an NVM-centric perspective, the fundamental limitation of this algorithm is that it is write-intensive, as it performs as many writes as the size of the input on every merge pass.

For sorting NVM-resident data, the database system can instead use a read-intensive sorting algorithm such as selection sort that involves taking multiple read passes over the input data but only writes back each element of the input once at its final location. The main limitation of this algorithm is that it suffers from higher time complexity than the external merge sort algorithm.

However, external merge sort and selection sort can be combined to create a write-limited yet efficient hybrid algorithm, called the segment sort algorithm. This algorithm sorts a segment of the input using the faster write-intensive external merge-sort algorithm, and sorts the rest of the input using the slower write-limited selection sort algorithm. The segment sort algorithm outperforms its underlying primitive algorithms by 30% and halves the number of memory writes to NVM.

Level 4.0 of the NVM evolution: protocol and algorithm redesign

The advent of NVM is one of the most significant architectural changes that systems designers will face in the foreseeable future. We believe that level 4.0 in the evolution of NVM programming involves redesigning the fundamental protocols and algorithms to fully leverage the unique characteristics of NVM.

About the authors: *Joy Arulraj is an Assistant Professor of Databaseology in the School of Computer Science at Georgia Tech. Spyros Blanas is an Assistant Professor in the Department of Computer Science and Engineering at The Ohio State University.*

not represent those of ACM SIGARCH or its parent organization, ACM.

CONTRIBUTE

Editor: Rajeev Balasubramonian

Associate Editor: Christina Delimitrou

Contribute to Computer Architecture Today

RECENT BLOG POSTS

ARCHIVES

October 2017 (6)
September 2017 (6)
August 2017 (2)
July 2017 (4)
June 2017 (6)
May 2017 (6)
April 2017 (4)
March 2017 (2)

TAGS

Academia Accelerators ACM SIGARCH Advice Architecture Benchmarks Cloud computing Conference Conferences Databases Datacenters Distributed Systems Diversity Emerging Technology FPGA Hardware Industry Interview ISCA Machine Learning Measurements Memory Mentoring Methodology Mobile Near Data Computing Networking non-volatile Operating Systems Opinion People Performance Persistent Policy Programmability Quantum Computing Review Reviewing Security Specialization Storage Systems Travel Virtual Meetings Vision

SUBSCRIBE

RSS SUBSCRIBE

Get E-Mail Alerts, enter your Email:

SUBSCRIBE »

JOIN US

Keep up-to-date with the latest technical developments, network with colleagues outside your workplace and get cutting-edge information, focused resources and unparalleled forums for discussions. Learn more...

ACM SIGARCH

SIGARCH serves a unique community of computer professionals working on the forefront of computer design in both industry and academia. It is ACM's primary forum to interchange ideas about tomorrow's hardware and its interactions with software.

ACM SIGARCH

SIGARCH serves a unique community of computer professionals working on the forefront of computer design in both industry and academia. It is ACM's primary forum to interchange ideas about tomorrow's hardware and its interactions with software.

About the Site

This site is maintained by volunteers working in many programs of ACM SIGARCH. We thank you for visiting! If you have questions about the site, please send a note to our content editor.