Designing SSD-Conscious Data Systems

Tarikul Islam Papon (Boston University, USA)

Solid-state drives (SSDs) have become the dominant secondary storage devices replacing their great ancestors hard-disk drives (HDDs). Unlike HDDs, SSDs exhibit two distinct characteristics: (i) access *concurrency*, allowing multiple I/O operations to run simultaneously and fully utilize the device bandwidth, and (ii) *read/write asymmetry*, where writes are slower than reads. Despite these, since most storage-intensive applications are primarily designed for HDDs, they do not consider concurrency and asymmetry hence having subpar performance. We propose the **Parametric I/O Model (PIO)** to capture the fundamental differences between traditional and modern storage devices: *concurrency* and *asymmetry* as parameters [1, 2]. PIO allows for faithful storage modeling that captures those two device-specific properties and allows the designing of storage-intensive algorithms tailored to the device at hand [5]. Using this novel storage model, we develop a new **asymmetry & concurrency-aware DBMS bufferpool manager** that utilizes the device concurrency to bridge the asymmetry [3]. We further present a **concurrency-aware graph engine** for out-of-core systems that enables efficient concurrent storage access [4].

ACE Bufferpool Manager. We focus on DBMS bufferpool since it is closely connected to the storage device, hence, better storage modeling has the potential to improve its overall performance. However, current approaches treat reads and writes equally without leveraging device concurrency. We refactor the bufferpool design space by decoupling the write-back policy from the eviction policy and propose an asymmetry/concurrency-aware bufferpool manager named **ACE that utilizes the underlying device concurrency to amortize the high asymmetric write cost** [3]. The write-back policy always writes multiple pages concurrently (utilizing the device's *write concurrency*), hence amortizing the write cost. The eviction policy evicts one or multiple pages at the same time to enable parallel prefetching, exploiting the device's *read concurrency*. A key advantage of ACE is that it can be integrated with any existing page replacement policy with low engineering effort, while, any prefetching technique can also be integrated, essentially allowing **any existing bufferpool manager to be augmented by our approach**. We implement several popular page replacement policies and their ACE counterparts in PostgreSQL and evaluate ACE's benefits using TPC-C and several microbenchmarks. The ACE counterparts of all policies lead to significant performance improvements, exhibiting **up to** $1.3\times$ **speedup for mixed TPC-C transactions** with a negligible increase in total disk writes and buffer misses. The concept of better storage modeling and leveraging device concurrency and asymmetry can also be extended to file systems, especially when supporting storage-intensive applications.

CAVE Graph Manager. Graph traversal operations, known for their random-access heavy pattern, can benefit from SSD concurrency by parallelizing node and edge accesses, effectively distributing the workload across the SSD's parallel architecture. Hence, to assess the impact of better storage modeling on random access-intensive applications, we next focus on developing an SSD-aware graph manager. Our goal is to parallelize graph traversal algorithms without changing their core properties while utilizing the underlying SSD concurrency. To achieve this, we identify two key ways (intra/inter-subgraph parallelization) to parallelize traversal algorithms based on the graph structure and algorithm. We propose an SSD-aware graph processing system, named **CAVE that can harness the concurrency of the underlying storage devices** [4]. Specifically, CAVE provides the necessary infrastructure to parallelize graph traversal algorithms when several independent vertex accesses can be performed in parallel. In essence, CAVE takes advantage of the availability of multiple paths that can be explored in parallel. CAVE uses a block-based file format based on adjacency lists, ensuring that graph metadata, vertex information, and edge information are stored in aligned blocks while enabling efficient support for graph traversal and analytical operations. Overall, **CAVE identifies independent storage accesses (thus parallelizable) and performs them concurrently based on the device's *optimal concurrency***. We develop in CAVE the parallelized versions of five popular graph algorithms: Breadth-First Search, Depth-First Search, Weakly Connected Components, PageRank, and Random Walk. We observe that CAVE can be up to **three orders of magnitude** faster than baseline GraphChi and up to **one order of magnitude** faster than GridGraph and Mosaic.

Overall, our experience shows that **incorporating asymmetry and concurrency in algorithm design leads to more faithful storage modeling and, ultimately, to better performance**.

References

[1] T. I. Papon, and M. Athanassoulis. "The Need for a New I/O Model," *CIDR*, 2021.

[2] T. I. Papon, and M. Athanassoulis, "A Parametric I/O Model for Modern Storage Devices," *DAMON*, 2021.

[3] T. I. Papon, and M. Athanassoulis, "ACEing the Bufferpool Management Paradigm for Modern Storage Devices," *ICDE*, 2023.

[4] T. I. Papon, T. Chen, S. Zhang, and M. Athanassoulis, "CAVE: Concurrency-Aware Graph Processing on SSDs," *SIGMOD*, 2024.

[5] T. I. Papon, "Enhancing Data Systems Performance by Exploiting SSD Concurrency & Asymmetry," *ICDE PhD Symposium*, 2024.