The Need for a New I/O Model

Tarikul Islam Papon
Boston University

Manos Athanassoulis
Boston University

Data-intensive applications performance is typically bounded by the time needed to transfer data through the storage and memory hierarchy. As a result, measuring and modeling *disk I/O accesses* is used as a proxy to performance. The traditional I/O model [1] considers a two-level memory hierarchy with a fast internal memory of bounded size (*memory*) and a slow unbounded external memory (*storage*), which are both divided into fixed-size blocks. Any computation requires to have the corresponding data blocks in memory. Accessing storage is typically orders of magnitude slower than accessing memory, thus, the traditional I/O model considers only storage accesses. This modeling approach closely describes reality when two key underlying assumptions hold: (i) disk reads and writes have similar cost, and (ii) applications can perform one I/O at a time. However, those two assumptions are *not* true for solid-state disks (SSD). The mismatch between the I/O model and contemporary devices is attributed to the fact that it was developed for hard disk drives (HDD). HDDs have *symmetric* read-write performance that is dominated by the seek time and rotational delay. The mechanical components of HDDs, further, do not allow them to serve multiple concurrent requests, since each request has to individually go through the costly mechanical movement.

Read/Write Asymmetry and Concurrency. A key property of SSDs (both off-the-shelf SATA, and high-end NVMe and PCIe devices) is the *erase-before-write*. To overwrite a previously written location, the corresponding location must first be erased before writing new data, leading to a significant **read-write asymmetry** [3]. The level of this asymmetry depends on the specific device as well as the type of access (sequential/random).

Another fundamental design property of SSDs is their *internal parallelism* in various levels (e.g., channel, chip, die, plane), which can be exploited to increase performance [2]. In other words, a device needs to receive multiple concurrent I/Os to reach its full bandwidth. The level of concurrency needed to saturate the device depends on the request type (read or write), as well as the properties of the specific device. Figure 1(A) shows the comparison of several recent Intel SSDs with respect to both their asymmetry and concurrency. The listed devices have asymmetry between 2× and 15.4× and concurrency between 7 and 18.

It is essential to know the level of parallelism of a specific device, which in turn enables servicing **concurrent** requests, otherwise the device may remain vastly underutilized. Figures 1(B) and 1(C) show the impact of concurrency as well as asymmetry. Specifically, Figure 1(B) shows the sustained read and write bandwidth when issuing requests on a PCIe SSD device using 1 and 8 threads. We observe that by increasing the concurrency from 1 to 8, there is a 6.6× increase in read bandwidth and 2× increase in write bandwidth. In other words, by matching the concurrency supported by the device (and hence using 8 concurrent threads) we fully utilize this device. Figure 1(C) depicts the impact of asymmetry and concurrency combined, when considering a specific storage access component of a Database Management System (DBMS). We focus

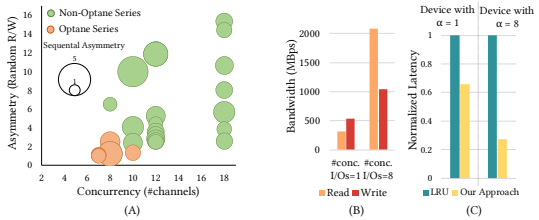

Figure 1: (A) *Asymmetry* and *Concurrency* in recent Intel SSDs. (B) Increasing *concurrency* increases device bandwidth. (C) Considering *asymmetry* helps significantly more.

on the bufferpool of a DBMS, which is the component that sits between the application (which is the SQL workload in this case) and the storage device. The figure shows the latency of executing a transactional workload in a bufferpool that uses LRU to evict page (green bars), and a new *asymmetry/concurrency-aware* eviction policy that batches write-back requests (from bufferpool to the disk) in a storage-aware manner (yellow bars). Both experiments are shown for a device with no asymmetry (behaving like a HDD) and a device with an asymmetry level of 8×. For each device, the latency is normalized with respect to LRU. We observe that our *asymmetry/concurrency-aware* algorithm has a 1.5× speedup for no asymmetry, while the benefit increases to 3.65× with asymmetry 8, because we treat reads and writes differently, by taking into account the different latency they have on the specific device.

The question we set out to answer is: *How should the I/O model be adapted in light of read/write asymmetry and concurrency?*

The Need for a New I/O Model. To answer this question, we propose a simple yet expressive storage model, that considers asymmetry (α) and concurrency (k) as parameters. This richer I/O model is able to capture contemporary state-of-the-art (and future) devices. By capturing α and k, we can make device-specific decisions at algorithm design time, rather than as an optimization during deployment and testing. Specifically, we envision better algorithm design for almost any component of a system that interacts with storage. For example, by making asymmetry and concurrency-aware decisions, a bufferpool will not trade one read for one write when it is saturated, rather it will attempt to prioritize multiple concurrent writes on the device and potentially some speculative concurrent reads. As another example, algorithms for tree and graph traversal will be able to access multiple nodes concurrently and avoid the classical sequential paradigm of accessing one node at a time, offering the same worst-case guarantees with faster search time on average. Overall, incorporating α and k in algorithm design allows for customizability for different devices which leads to *more faithful storage modeling* and, ultimately, to better device utilization.

REFERENCES

[1] Alok Aggarwal and Jeffrey Scott Vitter. The Input/Output Complexity of Sorting and Related Problems. *CACM* 31, 9 (1988).
[2] Feng Chen et al. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. *HPCA* (2011).
[3] Michael Cornwell. Anatomy of a solid-state drive. *CACM* 55, 12 (2012).