High Performance & Low Latency in Solid-State Drives Through Redundancy

Dimitris Skourtis, Scott Brandt, Carlos Maltzahn
University of California, Santa Cruz
{*skourtis, scott, carlosm*}*@cs.ucsc.edu*

Abstract

Solid-state drives are becoming increasingly popular in enterprise storage systems, playing the role of large caches and permanent storage. Although SSDs provide faster random access than hard-drives, their performance is especially workload-dependent and can be inconsistent to the point that it becomes worse than that of hard-drives (e.g., taking more than 100ms for a single read).

Many systems with mixed workloads have low latency requirements for reads and in such cases SSD inconsistencies become a problem. We present a solution that is based on redundancy and provides high performance, low latency and consistent read performance, as well as fault-tolerance, making it an alternative RAID-like design for solid-state drives.

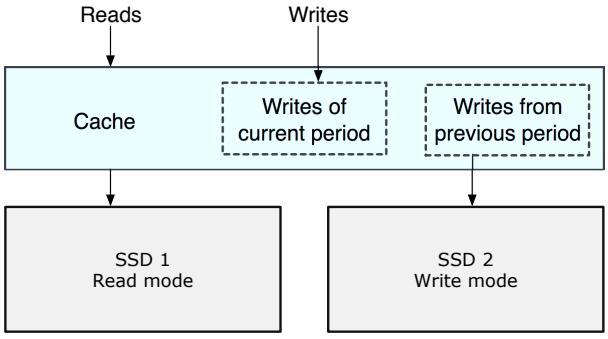


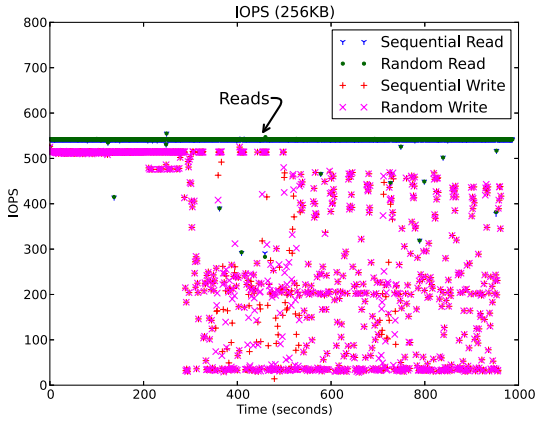
Figure 1: Each of the two drives is either performing reads or writes at any given time. While the first drive is reading, the other drive performs the writes of the previous period - before the drives switched roles.

Design and Evaluation

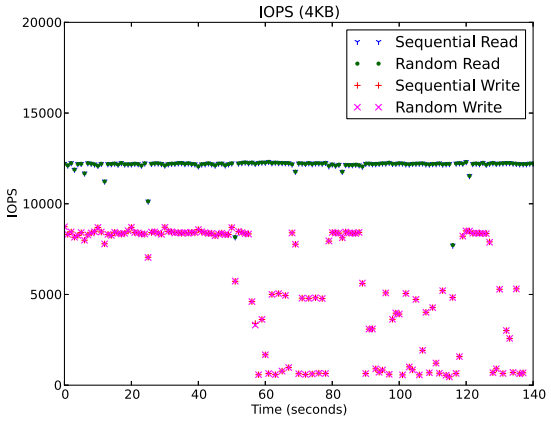
Solid-state drives have fast random access and often exhibit high performance. However, depending on their current and past workloads their performance can degrade quickly. For example, performing a mixture of large sequential and random writes over a large part of a drive's logical range can lead to high latencies. This is often due to the garbage collector not being able to keep up leading to what is known as internal fragmentation [1]. In such cases, mixing reads with writes leads to write-induced blocking events. Although such events may directly block less than 5% of the requests, they can account for a significant proportion of the device's time, e.g., 50%, leading to high latencies for queued requests.

We propose a new design based on redundancy that provides consistent performance and minimal latency for reads by physically isolating reads from writes. In other words we nearly eliminate the extra latency that reads have to pay due to writes, which is crucial for many applications with low latency requirements. Through that separation we minimize the read/write interference and have the opportunity to further optimize reads and writes separately. In terms of write performance, unless enough writes are synchronous or the system is allowed to be overloaded, the user conceives write performance as perfectly consistent. Note that using a single drive and dispatching reads and writes in batches as in [2], may improve the performance under certain workloads and SSD models. However, it does not eliminate the frequent blocking events under a generic workload.

Solid-state drive models differ from each other and a method or heuristic solution working on one SSD may not work equally well on another. We are interested in an approach that works across different models. Given two drives D_1 and D_2, we propose the separation of reads from writes by first sending reads to D_1 and writes to D_2. After a variable amount of time $T \geq T_{min}$, the drives switch roles with D_2 performing writes and D_1 reads. When the switch takes place, D_2 first performs all the writes D_1 completed that D_2 has not, so that the drives are in sync. If D_2 completes syncing and the window



(a) Under large requests, the read performance is high and consistent.



(b) Small requests also achieve consistent and high read performance.

Figure 2: By physically separating reads from writes we achieve high performance, consistency, and minimal latency for both small and large requests.

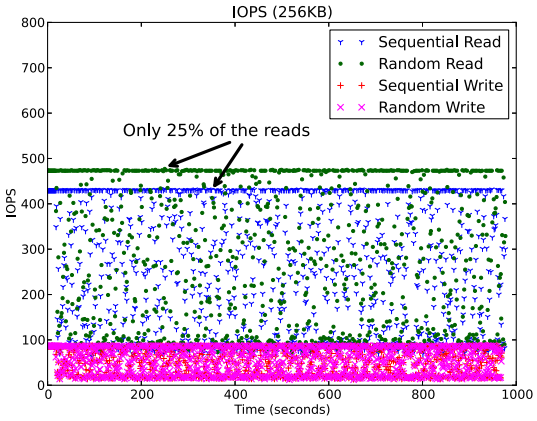


Figure 3: Using a single drive, the garbage collector cannot keep up, leading to prohibitively high read latencies and inconsistent performance. Our design solves this problem while doubling the read throughput.

is not yet over ($t < T_{min}$,) D_2 continues with new writes until $t \geq T_{min}$. In order to enable the above, we place a cache on top of the drives. All new writes first go to the cache, while the write-drive D_w is sent the writes that were stored in the cache during the previous window. Of course, if there are not enough writes in the cache after the drives are in sync, D_w may be used to increase the read performance. Finally, at any point in time, the union of the cache with any of the two drives contains exactly the same data. Hence, if one drive fails, no data is lost.

We implemented our design and verified that it provides high performance, low latency and read performance consistency under mixed workloads. For our experiments we used two 250GB Intel 510 SSDs that were previously written randomly. Note that although our design includes a cache and therefore cache hits as well as overwriting data still in cache are possible optimizations we looked at the worst-case and ignored potential cache hits. Our workloads consist of either small or large requests of four types, as shown in Figure 2. From the same figure we see that reads (1100 per sec) are not affected by writes, while writes have a similar performance as in our write-only experiments (not shown.) On the other hand, without redundancy reads achieve a total average of 525 reads/sec (Figure 3) when given 50% of the device time, which is less than half of our method. Most importantly the latencies are many times higher as well as inconsistent, while our approach achieves low read latencies consistently.

Besides offering low latency, our design makes it easier to provide efficient QoS for mixed workloads, and that is part of our current work. In addition, it can be combined with existing configurations such as RAID-01 and RAID-10. Finally, we are working on an evaluation under live workloads, such as databases and VMs.

References

[1] Chen, F., Koufaty, D. A., and Zhang, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems* (New York, NY, USA, 2009), SIGMETRICS '09, ACM, pp. 181–192.

[2] Park, S., and Shen, K. Fios: a fair, efficient flash i/o scheduler. In *Proceedings of the 10th USENIX conference on File and Storage Technologies* (Berkeley, CA, USA, 2012), FAST'12, USENIX Association, pp. 13–13.