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

Dimitris Skourtis, Dimitris Achlioptas, Carlos Maltzahn, Scott Brandt

Department of Computer Science
University of California, Santa Cruz
{skourtis,optas,carlosm,scott}@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 under read/write workloads is highly variable often exceeding that of hard-drives (e.g., taking 100ms for a single read). Many systems with mixed workloads have low latency requirements, or require predictable performance and guarantees. In such cases, the performance variance of SSDs becomes a problem for both predictability and raw performance.

In this paper, we propose a design based on redundancy, which provides high performance and low latency for reads under read/write workloads by physically separating reads from writes. More specifically, reads achieve read-only performance while writes perform at least as good as before. We evaluate our design using micro-benchmarks and real traces, illustrating the performance benefits of read/write separation in solid-state drives.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance

General Terms

Design, Performance

Keywords

Storage virtualization, solid-state drives, QoS, performance

1. INTRODUCTION

Solid-state drives have become an important component of many enterprise storage systems. They are commonly used as large caches and permanent storage, often on top of hard-drives, which operate as long-term storage. The main performance advantage of SSDs over hard-drives is their significantly faster random access. One would expect SSDs to be the answer to predictability, throughput and latency guarantees as well as performance isolation for consolidated storage in cloud environments. Unfortunately, their performance is heavily workload dependent. Depending on the drive and the workload there can be frequent and prohibitively high latencies up to 100ms for both writes and reads (due to writes) making SSDs multiple times slower than hard-drives in such cases. That type of interference has been noted in previous work [4, 5, 9, 11] for various device models and is well-known in the industry. Interference results in unpredictable performance and creates challenges in dedicated and especially in consolidated environments, where different types of workloads are mixed and clients require high throughput and low latency consistently, often in the form of reservations.

Although there is a continuing spread of solid-state drives in storage systems, research on providing high and stable performance for SSDs is limited. In particular, most related work focuses on performance characteristics [4, 5, 3], while other work, including [1, 2, 6] is related to topics on the design of drives, such as wear-leveling, parallelism and the Flash Translation Layer (FTL). With regards to scheduling, [11] provides fair-sharing while trying to improve the drive efficiency. To mitigate performance variance, current flash-based solutions for the enterprise are often aggressively over provisioned, costing many times more than commodity solid-state drives or otherwise offer lower write throughput. Given the fast spread of SSDs, we believe that providing stable performance and low latency efficiently is important for many systems.

In this paper, we propose and evaluate a method for achieving read-only throughput and latency for read requests under read/write workloads while providing write performance that is at least as good as before. More specifically, through a form of redundancy we physically separate reads from writes by presenting a subset of drives with read-only workloads and the rest of the drives with write-only workloads. After a certain amount of time the two sets switch roles with the write drives syncing up while also receiving new writes.

2. OVERVIEW

The contribution of this paper is a design based on redundancy that provides read-only performance for reads under arbitrary read/write workloads. In other words, we provide stable and high throughput at a high granularity, and minimal latency for reads while performing at least as good for writes as before. We present our results in three parts. In

INFLOW'13, November 3, 2013, Pennsylvania, USA.

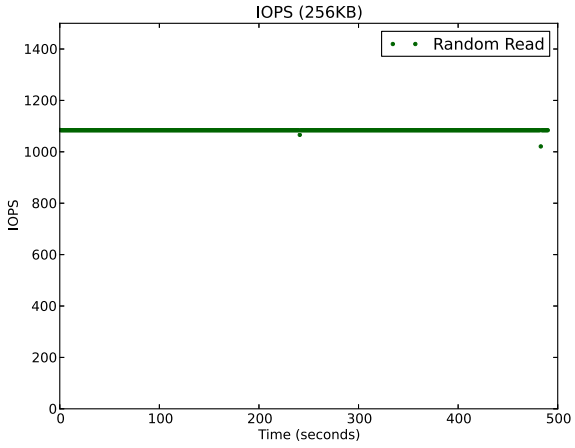


Figure 1: Read-only performance has virtually no variance.

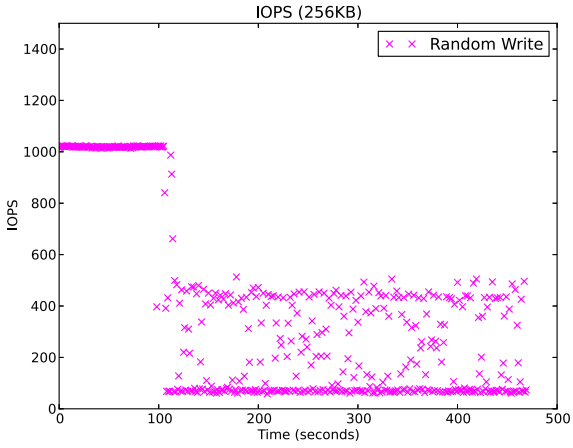


Figure 2: When the drive has limited free space, random writes trigger the garbage collector resulting in low and unstable performance.

Section 3, we study the performance of two SSD models. We observe their performance can become significantly unstable and that it depends on past workloads. As we are lowering the measurement granularity the worst-case throughput increases and stabilizes, which although expected, it is not obvious after which point the performance should become stable. Based on the above, in Section 4 we show how to provide read/write separation and evaluate our method under micro-benchmarks and real workload traces.

2.1 System Notes

For our experiments we perform direct I/O to bypass the OS cache and use Kernel AIO to asynchronously dispatch requests to the raw device. To make our results easier to interpret, we do not use a filesystem. Limited experiments on top of ext3 and ext4 suggest our method would work in those cases. Moreover, our experiments were performed with both our queue and NCQ (Native Command Queueing) depth set to 31. Other queue depths had similar effects to what is presented in [5], that is throughput increased with the queue size. Finally, the SATA connector was of 3.0Gb/s and used the following SSD models:

	Model	Type	Capacity	Cache	Year
A	Intel X-25E	SLC	65GB	16MB	2008
B	Intel 510	MLC	250GB	128MB	2011

We chose the above models to develop a method, which unlike heuristics (Section 3.2), works under different types of drives, including commodity drives. Recent, data-center oriented models have placed a greater emphasis on performance stability. For the drives we are aware of, the stability either comes at a cost that is multiple times that of commodity drives, or is achieved by lowering the random write throughput significantly compared to previous models.

3. PERFORMANCE AND STABILITY

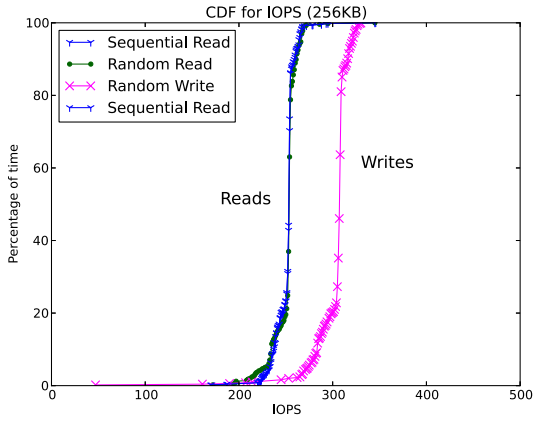
Solid-state drives have orders-of-magnitude faster random access than hard-drives. On the other hand, SSDs are stateful and their performance depends on past workloads. The interference of writes on reads has been noted in prior work [4, 5, 9, 11] for other drive models, and is widely known in the industry. We first verify the behavior of reads and writes on drive B. By running a simple read workload with requests of 256KB over the first 200GB of drive B, we see from Figure 1 that the throughput is virtually variance-free. We noticed a similar behavior for smaller request sizes and for drive A. On the other hand, performing the exact same experiment but with random writes gives stable throughput up to a point, after which the performance degrades and becomes unstable (Figure 2).

To illustrate the interference of writes on reads on the same drive, we run the following two experiments. In the first experiment, we have three streams performing reads and one performing random writes covering a logical range of only 100MB. The drive has over 200GB of free space. Figure 3(a) shows the CDF of the throughput in IOPS (input/output per second) achieved by each stream over time. We note that the performance behavior is relatively stable, which is expected, since more than 80% of the device blocks are clean, reducing write amplification. The second experiment consists of similar streams, but now the drive is filled and we perform writes uniformly at random over the first 200GB. Figure 3(b) illustrates that under such conditions the drive performance is unstable. We attribute this to the garbage collector not being able to keep up, which turns background operations into blocking ones.

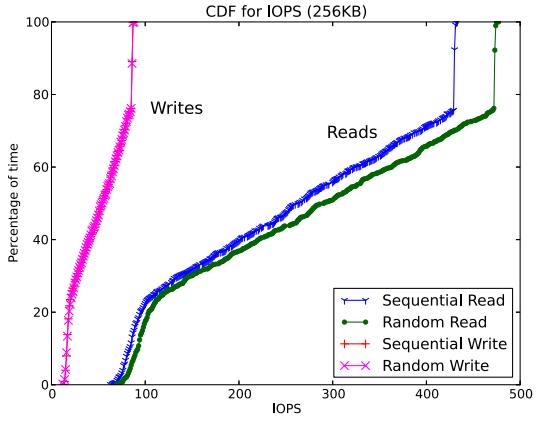
Different SSD models can exhibit different throughput variance for the same workload. Performing random writes over the first 50GB of drive A, which has a capacity of 65GB, gives a throughput variance close to that of reads (figure skipped). Still, the average performance eventually degrades to that of B and then, the total blocking time corresponds to more than 50% of the device time (Figure 4).

3.1 Performance Granularity

For certain workloads and depending on the drive, to maintain a throughput above zero the granularity has to be relatively low, i.e., multiple seconds. A simple example of such workload on drive A consists of 4KB sequential writes.



(a) Performing writes over only 100MB of a mostly empty SSD leads to stable write performance and the effect of writes on reads is relatively small.



(b) When the drive is filled and we perform writes over 200GB, the read throughput becomes unpredictable due to write-induced blocking events.

Figure 3: The performance variance varies depending on the writing range and the drive's free space.

We noted that when the sequential writes enter a randomly written area, the throughput oscillates between 0 and 40,000 writes/sec. We believe this variance is indeed due to the SSD itself rather than a system effect since disabling the drive write cache leads to stable but lower throughput of 2,500 writes/sec. In Figure 5, we see that the granularity has to be lowered to about 5 seconds before the variance becomes small. Note that the increase of the worst-case throughput is fast. Similarly, the worst-case throughput achieved by drive B when executing large sequential writes increases in the window size. Based on the above, it is safe to assume that the average write throughput of an SSD is stable enough over large time windows (e.g., 10 seconds). This observation will become important in Section 4.

To further emphasize the differences between SSD models with respect to throughput stability, we note that running 4KB sequential writes on drive B gives a stable performance (not shown). This implies that requests on drive B are not equally affected by the access patterns of previous writes,

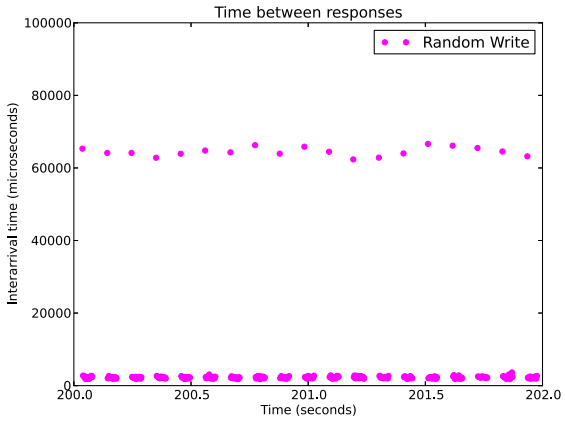


Figure 4: In each second, at least $10 \times 60ms$ are spent with the drive being blocked, leading to high latencies for all queued requests.

possibly due to its large cache. On the other hand, we saw that drive A provides a more stable performance for large random writes than B. The above imply that for mixtures of small sequential and large random writes, neither drive provides stable throughput at a granularity of a second. We conclude that stable performance at a high granularity depends on both the workload and the drive. Finally, the SSD write cache contributes to the throughput variance and although in our experiments we keep it enabled by default, disabling it improves stability but often at the cost of lower throughput, especially for small writes.

3.2 Heuristic Improvements

By studying drive models A and B, we found the behavior of A, which has a small cache, to be more easily affected by the workload type. First, writing sequentially over blocks that were previously written with a random pattern has a low and unstable behavior, while writing sequentially over sequentially written blocks has a high and stable performance. Although such patterns may appear under certain workloads and could be a filesystem optimization for certain drives, we cannot assume that in general. Moreover, switching from random to sequential writes on drive A, adds significant variance and lowers its performance.

To reduce that variance we tried to disaggregate sequential from random writes (e.g., in 10-second batches). Doing so doubled the throughput and reduced the variance significantly (to 10% of the average). On the other hand, we should emphasize that the above heuristic does not improve the read variance of drive B unless the random writes happen over a small range. This strengthens the position of not relying on heuristics due to differences between SSD models. In contrast to the above, in the next section, we present a generic method for achieving high and stable read performance under mixed workloads that is virtually independent of drive characteristics and past workloads.

4. HIGH AND STABLE PERFORMANCE

In the previous section, we observed that the granularity at which SSDs achieve high performance under mixed

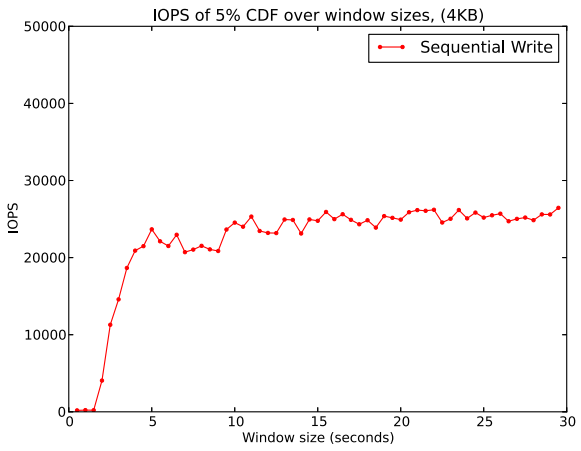


Figure 5: The worst-case throughput gets closer to the average as we increase the window over which we average. To reach 20,000 writes/sec we need windows of at least four seconds.

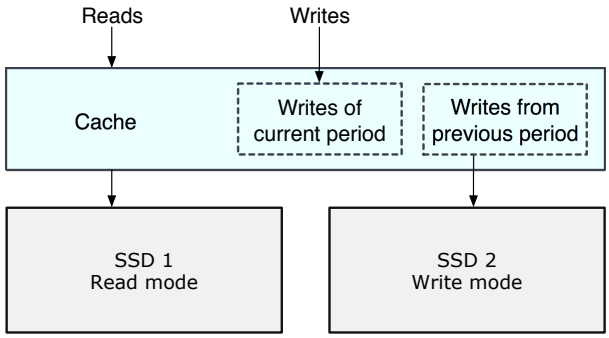


Figure 6: At any given time each of the two drives is either performing reads or writes. While one drive is reading the other drive is performing the writes of the previous time window.

(read/write) workloads is low, i.e., multiple seconds. That leads to high read latencies, which are prohibitive for many applications, as well as low throughput at a high granularity (Figure 5). We propose a generic design based on redundancy that when applied on SSDs provides predictable performance and low latency for reads, by physically isolating them from writes. We expect this design to be significantly less prone to differences between drives than heuristics, and demonstrate its benefits under two models.

In what follows, we first present a minimal version of our design, where we have two drives and perform replication. In Section 4.3, we generalize that to support erasure codes.

4.1 Basic Design

Solid-state drives have fast random access and can exhibit high performance. However, as shown in Section 3, depending on the current and past workloads, performance can degrade quickly. For example, performing random writes over a wide logical range of a drive can lead to high latencies for all queued requests due to write-induced blocking events (Figure 2). Such events can last up to 100ms and account for a significant proportion of the device's time, e.g., 60% (Figure 4). Therefore, when mixing read and write workloads, reads also block considerably, which can be prohibitive.

We want a solution that provides read-only performance for reads under mixed workloads. SSD models differ from each other and a heuristic solution working on one model may not work well on another. We are interested in an approach that works across models. We propose a new design based on redundancy that achieves those goals by physically isolating reads from writes. By doing so, we nearly eliminate the latency that reads have to pay due to writes, which is crucial for many low-latency applications. Moreover, we have the opportunity to further optimize reads and writes separately. Note that using a single drive and dispatching reads and writes in small time-based batches and giving priority to reads as in [11], may improve the performance under certain workloads and SSD models. However, it cannot eliminate the frequent blocking occurring due to garbage collection under a generic workload.

The basic design, illustrated in Figure 6, works as follows: given two drives D_1 and D_2, we separate reads from writes by 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_1 performing writes and D_2 reads. When the switch takes place, D_1 performs all the writes D_2 completed that D_1 has not, so that the drives are in sync. We call those two consecutive time windows a *period*. If D_1 completes syncing and the window is not yet over ($t < T_{min}$), D_1 continues with new writes until $t \geq T_{min}$. In order to achieve the above, we place a cache on top of the drives. While the writing drive D_w performs the old writes all new writes are written to the cache. In terms of the write performance, by default, the user perceives write performance as perfectly stable and half that of a drive dedicated to writes. In what follows we present certain properties of the above design and then a generalization supporting erasure codes and an arbitrary number of drives.

4.2 Properties and Challenges

4.2.1 *Data consistency & fault-tolerance*

By the above design, reads always have access to the latest data, possibly through the cache. This is because the union of the cache with any of the two drives always contains exactly the same (and latest) data. By the same argument, if any of the two drives fail at any point in time, there is no data loss and we continue having access to the latest data. In that case, the performance will be degraded until the failed drive is replaced and the data synced.

4.2.2 *Cache size*

Assuming we switch drive modes every T seconds and the write throughput of each drive is w MB/s, the cache size has to be at least $2T \times w$. This is because a total of $T \times w$ new writes are accepted while performing the previous $T \times w$ writes to each drive. We may lower that value to an average of $3/2 \times T \times w$ by removing from memory a write that is performed to both drives. As an example, if we switch every 10 seconds and the write throughput per drive is 200MB/s, then we need a cache of $T \times 2w = 4000$MBs.

The above requires that the drives have the same throughput on average (over T seconds), which is reasonable to assume if T is not small (Figure 5). Moreover, we assume that the rate at which writes are accepted is $w/2$, i.e., half the write throughput of a drive, which is achieved through throttling. That condition is necessary to hold over large periods since replication implies that the write throughput, as perceived by the client, has to be half the drive throughput. Of course, extra cache may be added to handle bursts. Finally, the cache factor can become $w \times T$ by sacrificing up to half the read throughput if the syncing drive retrieves the required data from D_r instead of the cache. However, that would also sacrifice fault-tolerance and given the low cost of memory it may be an inferior choice.

4.2.3 Power failure

In an event of a power failure if the memory is volatile, then our design as described so far will result in a data loss of $T \times w$ MBs, which is less than 2GB in the above example. Shorter switching periods result in smaller data losses. However, we can avoid data loss by turning incoming writes into synchronous ones. Then the incoming $T \times w/2$ MBs of data is written to D_w and the cache. After the power is restored, we know which drive has the latest data, as long as we store the mode each drive is in every T seconds (e.g., in a file). In that case, the cache amount required reduces to an average of $T \times w/2$. A disadvantage is that the write performance may be less stable than sending writes to the memory and only eventually committing to drive. Given the advantages of the original design, a small data loss may be tolerable by certain applications, especially in systems where recent data reside on multiple nodes. However, other applications may not tolerate a potential data loss.

Another approach to solve the power-failure problem while maintaining close to perfect write stability without assuming a battery-backed memory, is to perform writes to a permanent journal. Depending on how our design is used that could happen in different ways. First, in a distributed storage system each node often has a separate journal drive, which would be used by default. Second, on a local setup, a separate drive would have to be used. Since most nodes have a hard-drive, it could be used as a journal drive as the sequential performance of a hard-drive is comparable to that of an SSD, i.e., 100MB - which is the rate at which we accept writes at a stable state. Another approach, which is less flexible is to construct an SSD that would include what we logically think of as two, and in addition to that a small (e.g., 10GB) circular flash buffer to play the role of a journal. Finally, since no write is removed from the memory until it is performed on both drives, if our process fails but the system does not reboot, when the process restarts we may flush all writes to both drives.

4.2.4 Capacity and cost

Doubling the capacity required to store the same amount of data appears as doubling the storage cost. However, there are reasons why this is not entirely true. First, cheaper SSDs may be used in our design because we are taking away responsibility from the SSD controller by not mixing reads and writes. In other words, any reasonable SSD has high and stable read-only performance, and stable average write performance over large time intervals. Second, applications requiring high throughput (at low granularities) and low latency

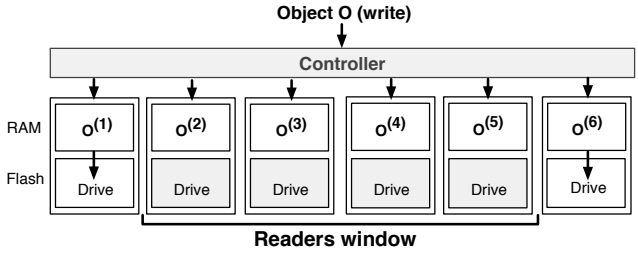


Figure 7: Each object is obfuscated and its chunks are spread across all drives. Reading drives store their chunk in memory until they become writers.

lead to over-provisioning due to the unstable performance of mixed workloads. Third, providing a drive with a write-only drive for multiple seconds instead of interleaving reads and writes is expected to improve its lifetime. Finally, most distributed storage systems already employ redundancy so the hardware is available for fault-tolerance and only the software has to be adjusted to apply this design. In the next section, we reduce the storage space penalty through erasure codes while providing read/write separation.

4.3 Design Generalization

We now describe the generalization of the previous design to support an arbitrary number of identical SSDs and reader-to-writer drive ratios through erasure coding.

Assume a system with N drives connected over the network to a single controller. We want to build a fault-tolerant storage system by using N identical solid-state drives. We will model redundancy as follows. Each object O stored in our system will occupy $q|O|$ space, for some $q > 1$. Having fixed q, the best we can hope in terms of fault-tolerance and load-balancing is that the $q|O|$ bits used to represent O are distributed (evenly) among the N drives in such a way that O can be reconstituted from any set of N/q drives. A natural way to achieve load-balancing within each group is the following: To handle a write request for an object O, each of the N drives receives a write request of size $|O| \times q/N$. To handle a read request for an object O, each of N/q randomly selected drives receives a read request of size $|O| \times q/N$.

In the system above, writes are load-balanced deterministically since each write request places exactly the same load on each drive. Reads, on the other hand, are load-balanced via randomization. Each drive receives a stream of read and write requests whose interleaving mirrors the interleaving of read/write requests coming from the external world (more precisely, each external-world write request generates a write on each drive, while each external-world read request generates a read with probability $1/q$ on each drive.)

As discussed in Section 3, in the presence of read/write interleaving the write latency "pollutes" the variance of reads. We would like to avoid this latency contamination and bring read latency down to the levels that would be experienced if each drive was read-only. To this effect, we propose making the load-balancing of reads partially deterministic, as follows. Place the N drives on a ring. On this ring consider a sliding window of size s, such that $N/q \leq s \leq N$. The window moves along the ring one location at a time at a constant speed, transitioning between successive locations

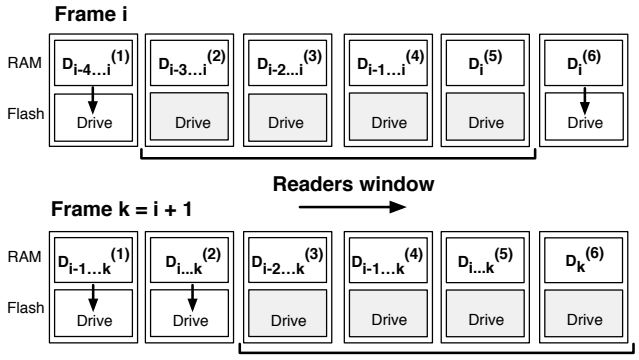


Figure 8: Each node m accumulates the incoming writes across frames $f \ldots f'$ in memory, $D^{(m)}_{f \ldots f'}$. While outside the reading window nodes flush their data.

"instantaneously". The time it takes the window to complete a rotation is called the period P. The amount of time, P/N, that the window stays in each location is called a frame.

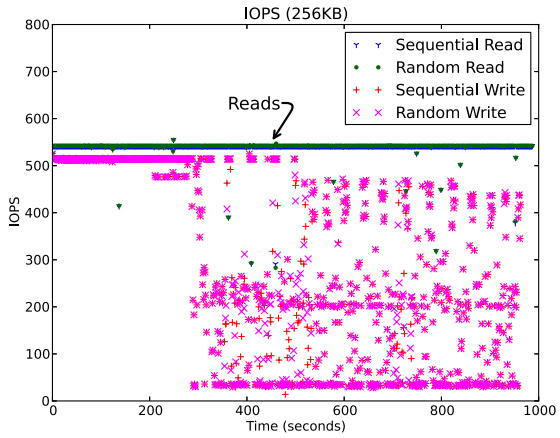
To handle a write request for an object O, each of the N drives receives one write request of size $|O| \times q/N$ (Figure 7, with $N = 6$ and $q = 3/2$). To handle a read request for an object O, out of the s drives in the window N/q drives are selected at random and each receives one read request of size $|O| \times q/N$. In other words, the only difference between the two systems is that reads are not handled by a random subset of nodes per read request, but by random nodes from a coordinated subset which changes only after it has handled a large number of read requests.

In the new system, drives inside the sliding window do not perform any writes, hence bringing read-latency to read-only levels. Instead, while inside the window, each drive stores all write requests received in memory (local cache/DRAM) and optionally to a log. While outside the window, each drive empties all information in memory, i.e., it performs the actual writes (Figure 8). Thus, each drive is a read-only drive for $P/N \times s \geq P/q$ successive time units and a write-only drive for at most $P(1-1/q)$ successive time units.

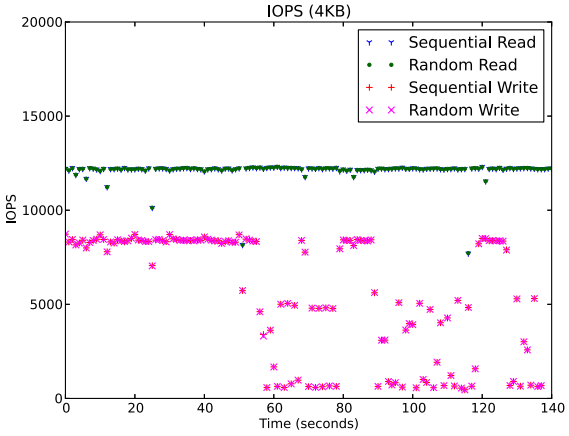
Clearly, there is a tradeoff regarding P. The bigger P is, the longer the stretches for which each drive will only serve requests of one type and, therefore, the better the performance (both in terms of throughput and in terms of latency). On the other hand, the smaller P is, the smaller the amount of memory needed for each drive.

Let us now look at the throughput difference between the two systems. The first system can accommodate any ratio of read and write loads, as long as the total demand placed on the system does not exceed capacity. Specifically, if r is the read-rate of each drive and w is the write-rate of each drive, then any load such that $R/r + Wq/w \leq N$ can be supported, where R and W are the read and write loads, respectively.

In the second system, s can be readily adjusted on the fly to any value in $[N/q, N]$, thus allowing the system to handle any read load up to the maximum possible rN. For each such choice of s, the capacity $N - s$ of the system provides write throughput, which thus ranges between 0 and $W_{\text{sep}} = w \times (N - N/q)/q = w \times N(q-1)/q^2 \leq wN/4$. As long as the write load does not exceed W_{sep} the system



(a) The read streams throughput remains constant at the maximum possible, while writes perform as before.



(b) The read throughput of 4KB requests remains stable at its maximum performance.

Figure 9: Physically separating reads from writes leads to a stable and high read performance.

perfoms perfect read/write separation and offers the read latency of a read-only system. We expect that in most shared storage systems, the reads-to-writes ratio and the redundancy are such that the above restriction is satisfied in the typical mode of operation. For example, for all $q \in [3/2, 3]$, having $R > 4W$ suffices.

When $W > W_{\text{sep}}$ some of the dedicated read nodes must become read/write nodes to handle the write load. As a result, read/write separation is only partial. Nevertheless, by construction, in every such case the second system performs at least as well as the first system in terms of read-latency.

4.3.1 *Feasibility and Efficiency*

We consider the following four dimensions: storage space, reliability, computation and read performance variance. Systems performing replication have high space requirements. On the other hand, they offer reliability, no computation costs for reconstruction while applying our method improves their performance variance without affecting the other quantities. In other words, the system becomes strictly better.

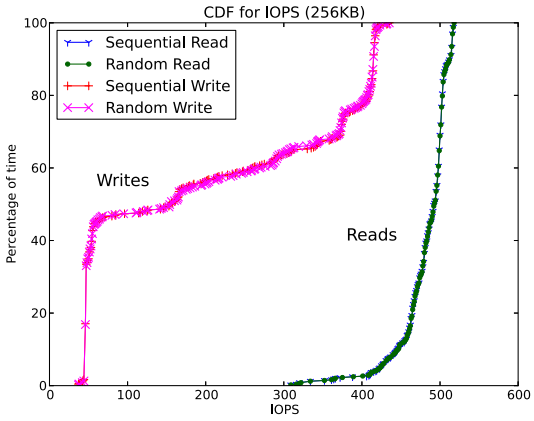


Figure 10: Using our design, the read throughput of drive A is mostly stable (on the right), with a small variance due to writes after each drive switch.

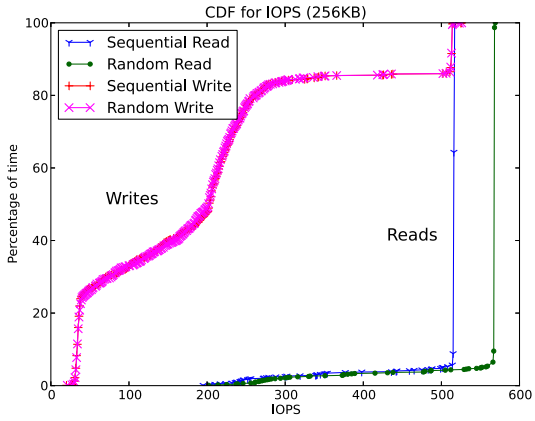


Figure 11: Using our design, on drive B we can provide predictable performance for reads more than 95% of the time without any heuristics.

Systems performing erasure coding have smaller storage space requirements and offer reliability, but add computation cost due to the reconstruction when there is a failure. Adding our method to such systems improves the performance variance. The price is that reading entails reconstruction, i.e., computation.

Nevertheless, there are improvements regarding reconstruction speed [12] while optimizing degraded read performance is a possibility, as was done in [8] for a single failure. In practice, there are SSD-based systems that perform reconstruction frequently to separate reads from writes, illustrating that reconstruction costs are tolerable when N is small (e.g., 6). We are interested in the behavior of such systems under various codes as N grows, and identifying the value of N after which read/write separation becomes inefficient due to excessive computation.

4.4 Preliminary Evaluation

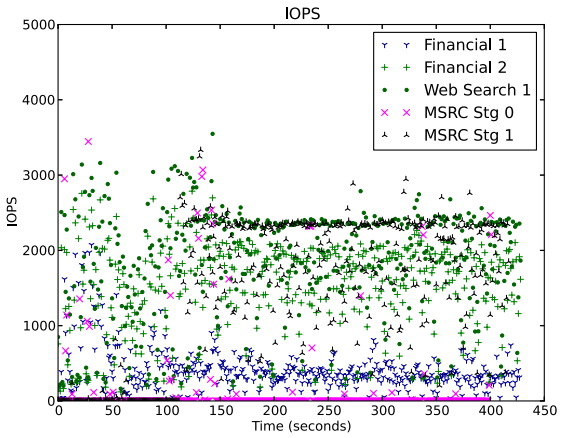
We built a prototype of our basic design as presented in Section 4.1 using replication and verified that it provides high performance, low latency and improved predictability for reads under mixed workloads. For simplicity, we ignored the possibility of cache hits or overwriting data still in the cache and focused on the worst-case performance. In what follows, the drives switch roles every $T_{min} = 10$ seconds. For the first experiment we used two instances of drive model B. The workload consists of large requests of all four types as shown in Figure 9(a). From the same figure, we see that reads happen at a total constant rate of 1100 reads/sec. and are not affected by writes. At the same time, writes have a variable behavior as in earlier experiments, e.g., Figure 2. On the other hand, as we saw earlier, on a single SSD reads have unstable performance due to the writes (Figure 3(b)).

Although we physically separate reads from writes, in the worst-case there can still be interference due to remaining background work right after the drives switch modes. In the previous experiment we noticed little interference and that was partly due to the drive itself. From Figure 10 we see that drive A performs significantly better than without redundancy, though reads do not appear as a perfect line, possibly due to its small cache. Since that may also happen with other drives we propose letting the write drive idle before each switch in order to reduce any remaining background work. That way, we found that when the drive starts reading, the interference is minimal and 99% of the time the throughput is stable. If providing QoS, that idle time can be charged to the write streams, since they are responsible for the blocking. Having small amounts of interference may be acceptable in many cases, however, certain users may be willing to sacrifice part of the write throughput to further reduce the chance of high read latency.

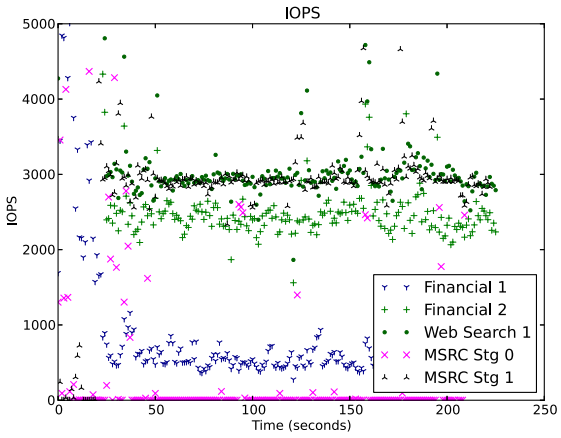
4.5 Evaluation with Traces

In this section we include a short evaluation of our design using the Stg dataset from the MSR Cambridge Traces [10], OLTP traces from a financial institution and traces from a popular search engine [14]. Other combinations of MSRC traces gave us similar results with respect to read/write isolation and skip them. For the following experiments we used drive model B and performed large writes before running the traces to fill the drive cache. Evaluating results using traces can be more challenging to interpret due to request size differences leading to a variable throughput even under a storage system capable of delivering perfectly stable performance.

In terms of read/write isolation, Figure 12(a) shows the high variance of the read throughput when mixing reads and writes under a single device. Under the same workload our method provides stable performance (Figure 12(b)). The write plots for both cases are skipped as they are as unstable as 12(a). Figure 13(a) focuses on twenty arbitrary seconds of the same experiment and illustrates that read/write interleaving leads to response times in the range of 100ms. When a request blocks, all streams get blocked leading to a lower and variant throughput as illustrated earlier. Looking more closely, we see that about 25% of the time reads are blocked due to the writes. On the other hand, from Figure 13(b) we see that our method provides read-only response time that is low and stable, almost always.

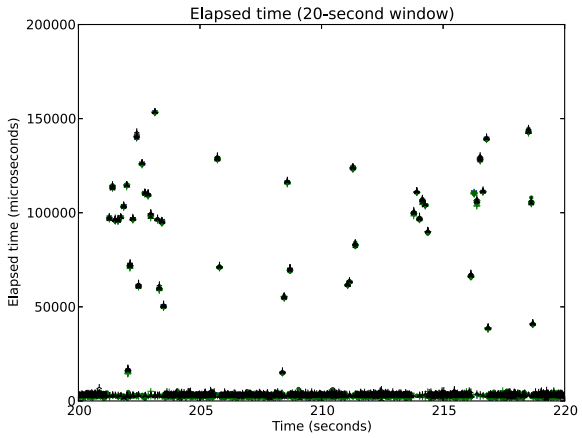


(a) Under a single drive, reads are blocked by writes (not shown) making read performance unpredictable.

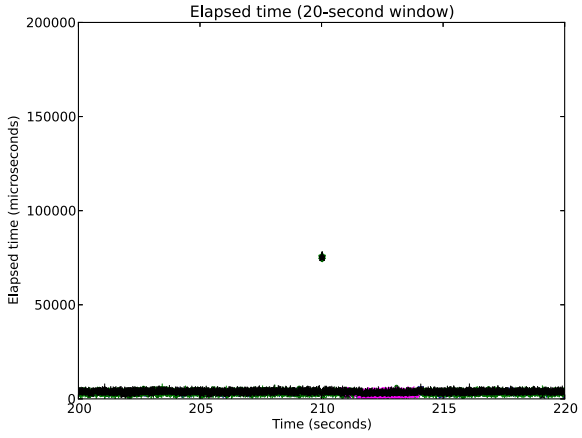


(b) Under our design, reads are not affected by writes and the same amount of reads completes in half the time.

Figure 12: Under a mixture of real workloads our method stabilizes the read performance.



(a) Under a single drive, the garbage collector blocks reads for tens of milliseconds, or for 25% of the device time.



(b) Under the proposed design, reads are virtually unaffected by writes - they are blocked for less than %1 of the time.

Figure 13: High-latency events become rare when physically separating reads from writes.

5. RELATED WORK

Although there are a number of papers on the performance and characteristics of SSDs, there is little work taking advantage of such results in the context of scheduling and performance consistency. An example of a fair scheduler optimized for flash is FIOS [11]. According to the results in [11], FIOS is an improvement over OS schedulers that are designed with hard-drive characteristics in mind. FIOS is not designed to provide read-only performance but rather as an efficient flash scheduler. We are interested in minimal read latencies that are significantly less than 100ms. In another direction, SFS [9] presents a filesystem designed to improve write performance by sequentializing writes.

A number of papers study the performance characteristics of SSDs. [4] includes a set of experiments on the effect of reads/writes and access patterns on performance. [5] shows the effect of parallelism on performance, while [3] presents a benchmark and illustrates flash performance patterns. In addition, [13] presents system-level assumptions that need to be revisited in the context of SSDs. Other work focuses on design improvements. For example, [1] touches on a number of aspects of performance such as parallelism and write ordering. [6] proposes a solution for write amplification, while [2] focuses on write endurance and its implications on disk scheduling. Moreover, [7] focuses on the future of flash and the relation between its density and performance.

6. CONCLUSIONS

The performance of SSDs degrades and becomes significantly unstable under demanding read/write workloads. In this paper, we introduced a design based on redundancy that physically separates reads from writes to achieve read-only performance under mixed workloads. Through experiments, we demonstrated that under replication, our design enables stable and high performance for reads under read/write workloads. For future work, we plan to study its scalability using erasure codes, as well as its generalization to large-scale distributed flash-only storage systems.

7. REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[2] S. Boboila and P. Desnoyers. Write endurance in flash drives: measurements and analysis. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.

[3] L. Bouganim, B. ?Ůr JŮnsson, and P. Bonnet. uflip: Understanding flash io patterns. In *CIDR 2009, FOURTH BIENNIAL CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH*. CIDR, 2009.

[4] F. Chen, D. A. Koufaty, and X. Zhang. 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*, SIGMETRICS '09, pages 181–192, New York, NY, USA, 2009. ACM.

[5] F. Chen, R. Lee, and X. Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 266–277, Washington, DC, USA, 2011. IEEE Computer Society.

[6] P. Desnoyers. Analytic modeling of ssd write performance. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pages 14:1–14:10, New York, NY, USA, 2012. ACM.

[7] L. M. Grupp, J. D. Davis, and S. Swanson. The bleak future of nand flash memory. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[8] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 20–20, Berkeley, CA, USA, 2012. USENIX Association.

[9] C. Min, K. Kim, H. Cho, S. Lee, and Y. Eom. Sfs: Random write considered harmful in solid state drives. In *FAST'12: Proceedings of the 10th Conference on File and Storage Technologies*, 2012.

[10] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: practical power management for enterprise storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 17:1–17:15, Berkeley, CA, USA, 2008. USENIX Association.

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

[12] J. Plank, K. Greenan, and E. L. Miller. Screaming fast galois field arithmetic using intel simd extensions. In *Proceedings of the 11th Conference on File and Storage Systems (FAST 2013)*, Feb. 2013.

[13] A. Rajimwale, V. Prabhakaran, and J. D. Davis. Block management in solid-state devices. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 21–21, Berkeley, CA, USA, 2009. USENIX Association.

[14] OLTP traces from the University of Massachusetts Amherst trace repository. http://traces.cs.umass.edu/index.php/Storage.