Ianus: Guaranteeing High Performance in Solid-State Drives

Dimitris Skourtis
skourtis@cs.ucsc.edu

Scott Brandt
scott@cs.ucsc.edu

Carlos Maltzahn
carlosm@cs.ucsc.edu

Department of Computer Science
University of California, Santa Cruz

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 to the point that it becomes worse than that of hard-drives (e.g., taking 100ms for a single read). Many systems with read/write workloads have low latency requirements, or require predictable performance and guarantees. In such cases SSD performance variability becomes a problem for both predictability and raw performance.

First, we show how to provide tight throughput guarantees to multiple clients sharing the same solid-state drive storage. Second, we introduce and evaluate Ianus, a design based on redundancy, which achieves high performance, low latency and stable read throughput, as well as fault-tolerance, making it an alternative RAID-like design for solid-state drives. Finally, by combining the two solutions we provide QoS while maintaining high performance.

Categories and Subject Descriptors

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

Keywords

Storage virtualization, solid-state drives, QoS, resource allocation, performance

1. INTRODUCTION

Over the past few years, solid-state drives have become an important component of many enterprise storage systems. SSDs are currently used as large caches and permanent storage, often on top of hard-drives, which operate as long-term storage. The main advantage of SSDs over hard-drives is their significantly faster random access. One would expect SSDs to be the answer to performance predictability, throughput and latency guarantees as well as performance isolation between clients sharing the same drive or storage system in a cloud environment. Unfortunately, their performance is heavily workload dependent. Depending on the drive and the workload there can be frequent and prohibitively high latencies in the order of 100ms for both writes and reads (due to writes) making SSDs multiple times slower than hard-drives in such cases. This 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 QoS 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, [12] provides fair-sharing while trying to improve the drive efficiency. However, it does not attempt to provide QoS and is still susceptible to high latencies due to the nature of SSDs. Given the fast spread of SSDs, we believe that providing QoS and improving their performance stability is important for many systems.

Ideally we would like a solution providing high performance guarantees and low latency under any workload and drive. We consider guarantees and performance as orthogonal goals each of which requires an independent solution. In many cases, such as in shared storage or when applications have jobs with deadlines, it is important for a client to know what performance to expect in the worst case. Whether that performance is considered high or not is an independent matter. Therefore, in terms of providing guarantees we may ignore whether the worst-case performance of the typical SSD might be less than the hoped one and instead focus on achieving the promised performance, which largely depends on the device itself. In this paper, we provide tight guarantees through time-based scheduling, without the need of heuristic improvements.

The other dimension of the problem is providing high performance and low latency consistently. Towards that goal we need to find ways to improve the solid-state drive performance and its stability without necessarily modifying the drive itself. Since the behavior of different SSD models may differ, it is hard to ensure that a method or heuristic enhanc-

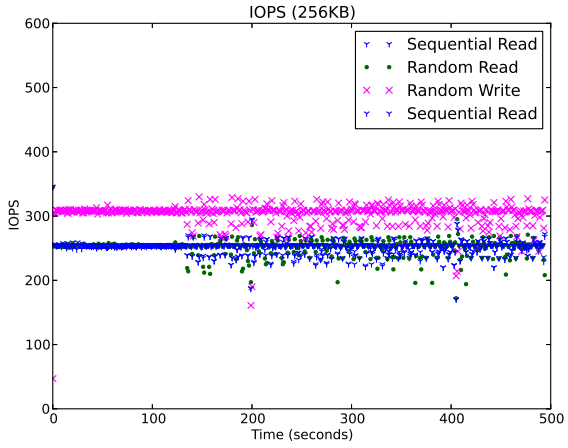


Figure 1: Performing writes over only 100MB of a mostly empty SSD leads to stable write performance and the effect of writes on reads is relatively small.

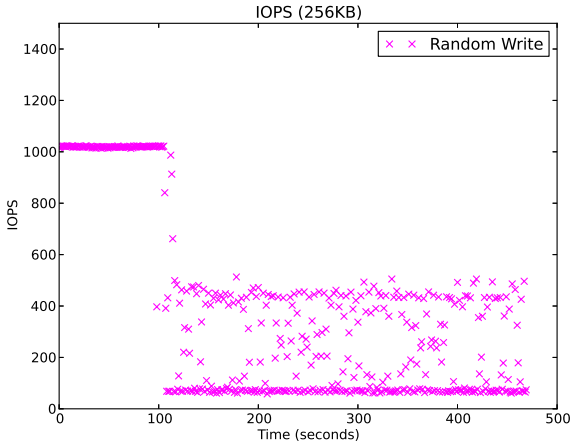


Figure 2: When the drive is already filled up, the garbage collector cannot keep up leading to blocking events and unpredictable performance.

ing the performance of a specific drive model will perform equally well (if not worse) on a different one. Instead, we note that in general, mixing reads with writes in commodity drives can lead to significant latencies in the order of 100ms due to the writes. That type of interference has been noted in previous work [4, 5, 10, 12] and is well-known in the industry. Such blocking events are responsible to a large degree for the low or variable performance a user eventually perceives when using SSDs. Given the above, we propose the physical separation of reads from writes through redundancy in order to consistently achieve low latency and high read throughput under read/write workloads. By combining the above with our method for performance guarantees, we can guarantee high read performance and low latency under mixed workloads.

2. OVERVIEW

The contribution of this paper is (1) *a method for providing high performance guarantees at different time granularities for shared solid-state drives*, as well as (2) *a design based on redundancy that stabilizes read throughput and latency under read/write workloads*. Each component can be used independently or under a single solution. We implemented a prototype of the above and present our results in three parts. First, in Section 3 we study the performance of two SSD models with different behavior. We observe that although SSD performance depends on past workloads and that it can become significantly unstable, as we are lowering the measurement granularity the worst-case throughput increases and stabilizes. Second, based on that observation, in Section 4 we present a method based on profiling and time-based scheduling for providing throughput guarantees at different granularities to clients sharing the same drive. In that section, we are not attempting to improve the performance variance of the drive through heuristics, instead we provide tight guarantees based on its own capabilities. Third, having observed that SSD performance may vary significantly at a high granularity and therefore worst-case guarantees become relatively low, in Section 5 we introduce a new design based on redundancy. This design uses two drives to physically separate reads from writes allowing us to guarantee high read performance and low latency consistently, while providing fault-tolerance.

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. Our method can take into account different queue sizes (Section 4). Finally, the SATA connector was of 3.0Gb/s and we used the following models of Intel SSDs:

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

3. WORST-CASE PERFORMANCE

Solid-state drives can achieve orders-of-magnitude faster random access than hard-drives. On the other hand, SSDs are stateful and their performance depends on past workloads. This observation and the interference of writes on reads has been noted in prior work [4, 5, 10, 12], and is widely known in the industry. In what follows we illustrate that behavior by two experiments on drive B. In the first experiment, we have four streams of different type, the drive has over 200GB of free space and one of the streams performs random writes over a range of only 100MB. Figure 1 shows 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 there are clean blocks in the device, which reduces write amplification. In

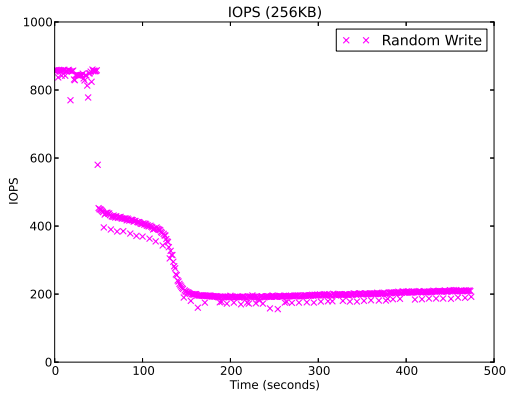


Figure 3: Starting with a sequentially written drive (A), random writes make it hard for the garbage collector to keep up, leading to low throughput.

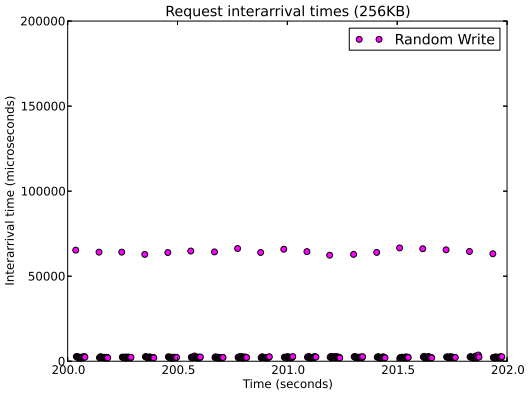


Figure 4: Within a second, ten high latency events occur, which in total make for half the drive time, leading to 200 rather than 400 IOPS in Figure 3.

the second experiment, the drive is filled and we perform writes uniformly at random over the first 200GB. Figure 2 illustrates that under such conditions the performance eventually degrades and becomes 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. Previously, we saw that random writes on drive B eventually exhibit variable and relatively low performance. Performing random writes over the first 50GB of drive A, which has a capacity of 65GB, gives a more stable throughput (Figure 3) than drive B. Still, the average performance eventually degrades to that of B. Moreover, the total blocking time corresponds to about 50% of the device time (Figure 4). The significance of the performance stability of drive A is that it enables higher guarantees at a granularity of a second. In what follows we study this observation in more detail, whereas in Section 4 we use the notion of granularity to enable higher throughput reservations.

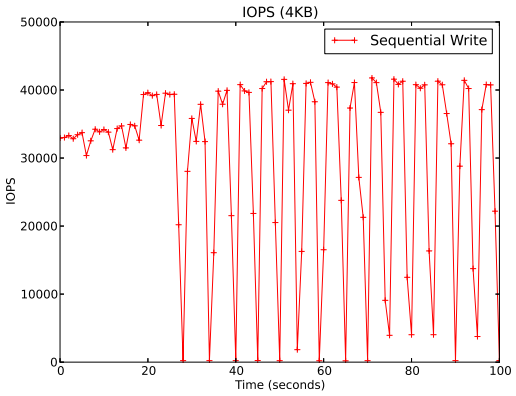


Figure 5: With the throughput of small writes oscillating from 0 to 40,000 IOPS, meaningful guarantees may only be provided at a low granularity.

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. From Figure 5, we note that after 20 seconds, when the sequential writes enter a randomly written area, the throughput oscillates between 0 and 40,000 writes/sec. Therefore, to provide meaningful guarantees we must lower the granularity. 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 6, we see that the granularity has to be lowered to about 5 seconds before the close to worst-case throughput is similar to the average. Also, we 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 (Figure 7).

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 as affected by the access patterns of previous writes, possibly due to its large cache. On the other hand, as shown before, 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. From the above, we conclude that stable performance at a high granularity depends on both the workload and the drive.

The above observations will be useful in the next section, where we present our method for providing performance guarantees. In particular, we will give clients the option of specifying the required granularity for their workload and trade it for higher throughput guarantees. Finally, note that the SSD write cache contributes to the throughput variance and although in our experiments we keep it enabled by default, disabling it does indeed improve stability but often at the cost of lower throughput, especially for small writes.

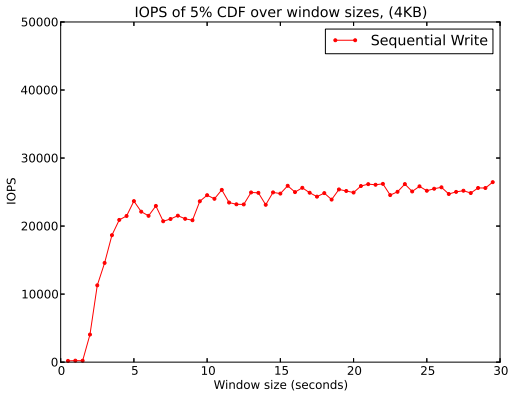


Figure 6: The reservable throughput increases with the window over which we average. To reach 20,000 writes/sec we need windows of at least four seconds.

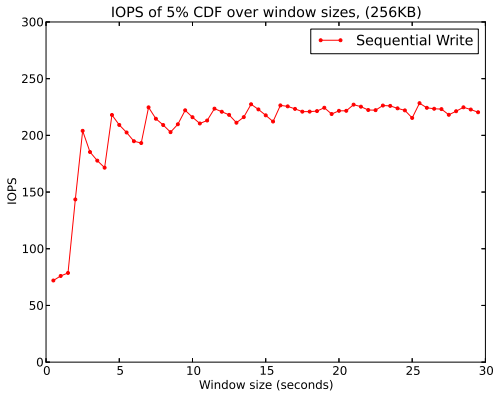


Figure 7: The throughput of the bottom 5% of sequential writes can be relatively low, making high granularity guarantees expensive.

4. PERFORMANCE GUARANTEES

In this section, we present a method for providing throughput guarantees to SSDs. To construct a generic QoS method, we distinguish between performance guarantees and improvements, e.g., through heuristics. In this section we focus on guarantees without attempting to improve the performance or its variance. Instead, we rely on the device capabilities. Despite that, we consider both goals as important and in Section 5 we introduce a design consisting of two SSDs that improves read performance stability under mixed workloads.

4.1 Overview

Storage users want Service Level Agreements (SLAs) in the form of throughput or latency requirements. Our solution supports throughput guarantees at different granularities for shared SSDs with each client receiving the promised performance irrespectively of other workloads. What follows is an overview of the steps we take to achieve that. First, we profile the drive by running simple workloads. Using the profiling results we construct throughput distributions allowing us to estimate the cost of each type of request at any granularity and confidence level. Based on those distributions, we convert the client requirements into utilization, which is defined as the fraction of device time provided to a client/workload. Afterwards, each request is assigned a deadline based on its cost estimate and the utilization assigned to its workload. Finally, each request is dispatched based on its deadline and according to the Earliest Deadline First algorithm. In the rest of this section, we describe the above in more detail and present our evaluation results.

4.2 Scheduling for Guarantees

In shared environments, where multiple streams utilize the same storage system, each stream expects its performance objectives to be satisfied independently of other workloads. Hence, any interference between different types of requests due to request cost inequalities has to be minimized and the utilization provided to each stream adjusted accordingly. To achieve that we follow a time-based approach [13] and provide each stream with a proportion of the drive time based on its requests and performance needs. By reserving time we can guarantee an absolute throughput for each stream rather than a proportion of the total throughput, which can vary significantly and lead to low reservations.

In order to reserve a specific percentage of the device time for each stream, we need to know the cost of the queued requests. The amount of time it takes for a request to complete, irrespectively of any queueing time varies, and estimating it online is a separate challenge [16]. From our experiments and from previous work [5] we know that in SSDs the request size and the queue depth are significant factors in a request's cost. Moreover, it is known from [4], and verify it here, that whether a request is a read or write, and whether it is sequential or random can significantly affect its cost. We conclude that by ignoring the state of an SSD, it is possible to estimate the cost of a request to a large degree based on its size, type and the queue depth. On the other hand, SSDs are stateful and their behavior is shaped by current and previous workloads. Hence, depending on the state of the drive, e.g., whether it has enough free space, the cost of a request may differ. Therefore, to provide close to worst-case guarantees we should consider a worst-case behavior.

To estimate request costs we profile the drive under requests of different access type, size and queue sizes. Since the throughput during profiling may fluctuate significantly, instead of taking the average cost, each stream has a confidence level assigned to its reservations, with higher levels implying a higher request cost. On a similar note, if the profiling range is small enough (e.g., 1GB,) the write performance may be more consistent as well as higher and the guarantees granularity can be higher. Having said that, the write range is not part of our model. Instead, if the storage user expects to consistently have a lighter workload it is up to them to adjust the profiling accordingly. If the workload is not known, then the profiling should cover the worst case as described in the previous section.

Although scheduling by time allows us to minimize stream interference and provide tighter guarantees, clients expect throughput reservations expressed by SLAs. To that end, we

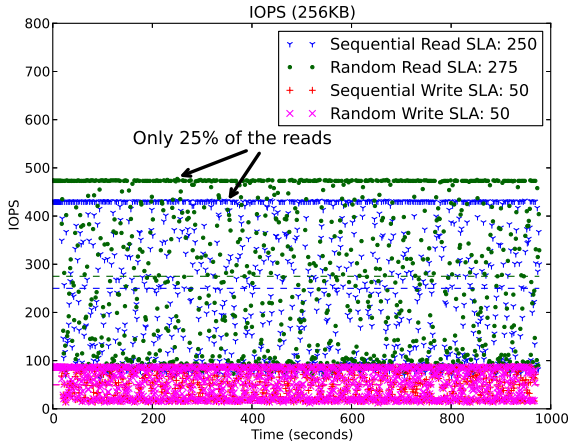


Figure 8: Due to write-induced blocking events, read throughput is unpredictable at a 1-second granularity and the targets are only met on average.

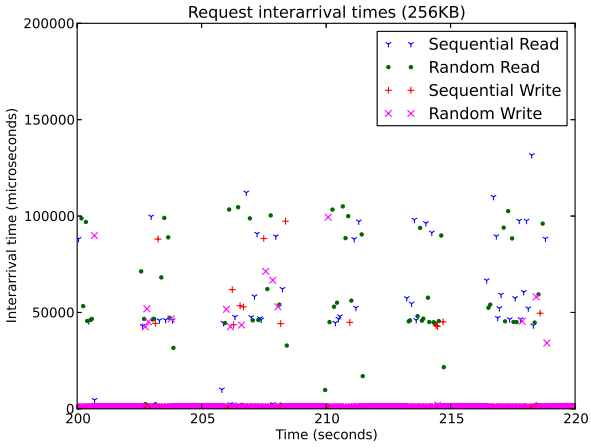


Figure 9: The blocking events do not have a short period, meaning that some, e.g., 1-second intervals have no or few such events while others have many.

use our profiling results to convert throughput requirements to the corresponding stream utilization u_s, which is defined as the proportion of the device time provided to stream s. Note that throughput requirements need to have a granularity associated with them and u_s depends on that value. In particular, a higher granularity is expected to imply a lower throughput (Figures 6 and 7) and a higher utilization. The notion of granularity is useful because there are applications such as video streaming, which would trade granularity for higher throughput guarantees, whereas online processes such as audio streams would accept lower throughput guarantees for higher granularity. By converting throughput to utilization we can also perform admission control, since the total utilization cannot exceed 100%. On the other hand, for systems that can be over-utilized, where guarantees are less important, this model offers stream isolation by providing each stream with its relative proportion of the device time.

In order to schedule the stream requests, given the expected cost of a request we assign it a deadline that depends on the time utilization rate u_s of the corresponding stream s. In particular, if the expected cost of a request is e, then its (relative) deadline is defined as $d = e/u$. Also, the absolute deadline of the request is set to $D_s = T_s + d$, where T_s is the sum of all relative deadlines of stream s up to that point. After a request is assigned a deadline, it is inserted to a priority queue and dispatched according to the Earliest Deadline First algorithm. To summarize, at a high level our method consists of the following sequence of operations: drive profiling, throughput requirements to utilization conversion, (i.e., resource allocation) and time-based request dispatching.

4.3 Providing Guarantees

In the previous section, we presented worst-case scenarios, where random writes span most of the logical space and SSD performance degrades significantly. Here, we apply our model to provide guarantees to multiple streams sharing the same drive and illustrate the importance of associating granularity to throughput guarantees. A natural question is how to decide the granularity at which we can provide certain guarantees. Since drives often behave differently, we take advantage of the profiling we already perform. For example, in Figure 7, we saw that increasing the granularity, decreases the guarantees for large sequential writes (similarly for random ones). Based on that information we can convert throughput requirements of certain granularity to the required utilization u, and decide whether those requirements can be guaranteed, i.e., if $u + \sum_i u_i \leq 1$.

We start with a set of four streams of different type, each having a throughput requirement. In particular, we set the target of the two read streams to 250 and 275 IOPS and set the write targets to 50 IOPS each. Although the write targets might appear low, according to the profiling results (for drive B) in order to achieve those targets at a low granularity their streams have to occupy a total of 50% of the drive time. Instead, at a granularity of a second the utilization required just for the sequential writes is over 60% (as implied by Figure 7) for a total of over 170%. This is due to the significantly higher worst-case cost of writes compared to that of reads when performed in isolation (as in profiling). As expected, from Figure 8 we see that the throughput targets are not achieved at a 1-second granularity since the total utilization required exceeds 100%. On the other hand, the average throughputs still satisfy the targets but there is an oscillating behavior for all streams. Another way to visualize why a 1-second granularity leads to a high utilization is by noting the blocking events over a typical 2-second subinterval as shown in Figure 9. We observe that the number of blocking events per second varies, while it remains similar over 10-second intervals leading to a lower worst-case request cost. As expected, increasing the granularity to 10 seconds leads to a write cost drop and a required utilization of 99% to 100%. As illustrated in Figure 10, the achieved throughput at a 10-second granularity remains close to the targets of each stream. Finally, since the total utilization is between 99% and 100%, we conclude that almost all of the drive's time is successfully reserved, while achieving the

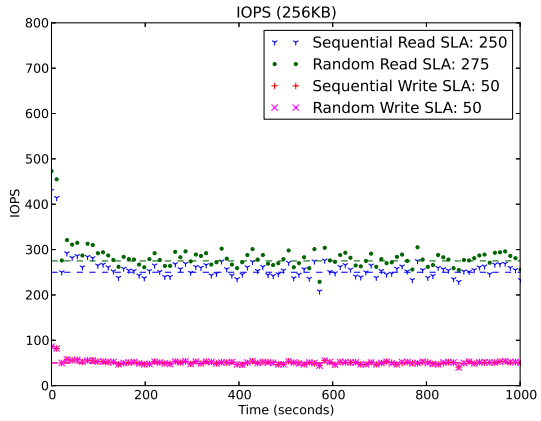


Figure 10: At a granularity of 10-seconds, the throughput is close enough to the targets while the reserved utilization is almost 100%.

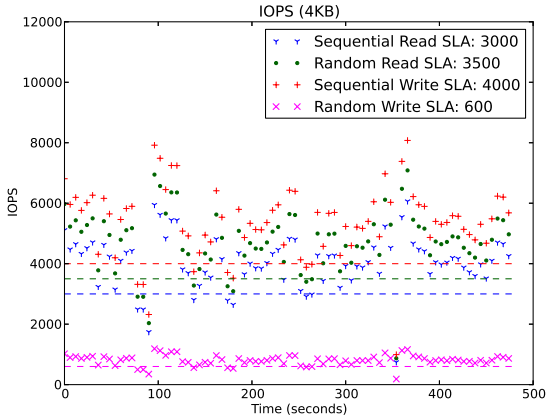


Figure 11: At a granularity of 5-seconds, the throughput guarantees are almost always achieved while the reserved utilization is almost 100%.

throughput guarantees to the granularity possible by drive B, without any heuristic optimizations. Moreover, this implies that the cost estimates extracted during profiling were accurate enough.

Large requests allow us to examine the drive behavior more easily by having a more consistent behavior than small (e.g., 4KB) requests. Although the behavior of small writes can be more variable, proper profiling and time scheduling still allow us to provide guarantees. As mentioned earlier, Figure 6 shows the relation between the throughput of sequential writes running in isolation on drive A and its granularity. We see that reducing the window size to less than four seconds drops the throughput quickly to the point where it becomes zero. In what follows we set the window size to five seconds, which corresponds to about 20,000 writes/second, therefore, an average cost of $50\mu s$ per request. From Figure 11, we see that the achieved throughput at a 5-second

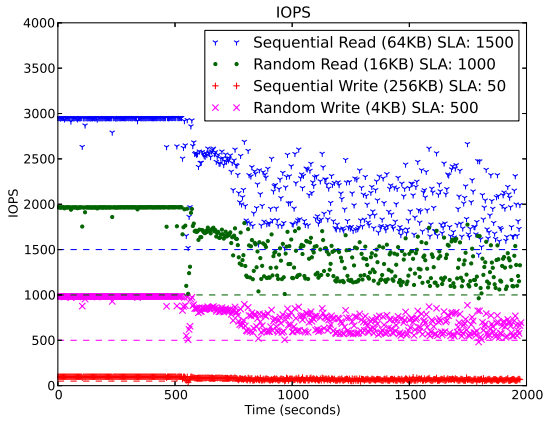


Figure 12: The guarantees (over 3-second intervals) for mixed request sizes are tightly satisfied, with a total device time reservation of 99%.

granularity meets the targets almost always, while the total reserved utilization reaches 100%. In the next experiment, as illustrated in Figure 12, we consider streams of different request sizes at a 3-second granularity, based on the granularity-throughput relation in Figure 13. Besides showing that the targets are met while reserving 99% of the device time, this experiment illustrates that a stream with larger requests can have a higher target than one with smaller requests without one making the other miss its targets. Moreover, since the scheduling happens based on the cost extracted from profiling, where each type of stream runs in isolation, we conclude that isolation is also provided to the granularity permitted by the device. Finally, our experiments so far were performed on a single SSD. We have noticed that on SSD arrays, dependencies between requests can lead to more frequent blocking. Therefore, to provide guarantees on arrays using the same method, the worst-case profiling has to be applied on top of the array.

4.4 Heuristic Performance 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 a pattern 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 effect 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), which increases our guarantees (figure skipped). 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, which strengthens the position of not relying on heuristics due to differences between SSD models. In contrast to the above, in the next

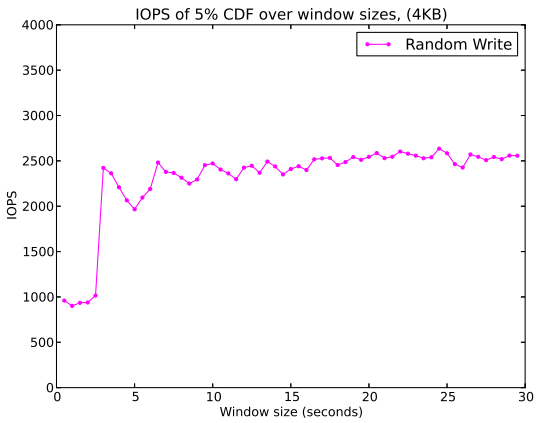


Figure 13: As long as we average over three seconds or more, the cost of a random write can be close to $400\mu s$. Otherwise, the cost doubles.

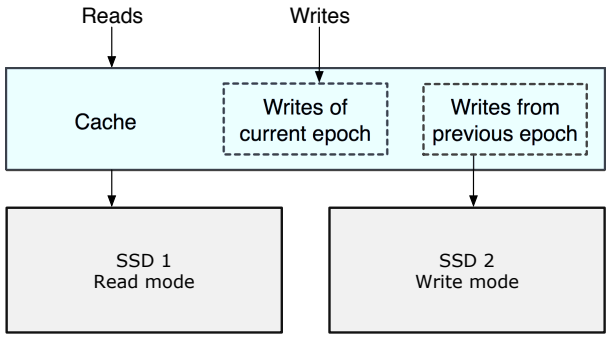


Figure 14: At any given time each of the two drives is either performing reads or writes. While one drive is reading the other drive is performing the writes of the previous epoch.

section, we present a method for providing high and stable read performance under mixed workloads for generic drives.

5. HIGH PERFORMANCE GUARANTEES

In the previous section, we observed that the granularity at which SSD devices achieve high performance under mixed (read/write) workloads is low, i.e., multiple seconds, instead of a single second or less. That leads to high read latencies, which are prohibitive for many applications. In this section, we propose a generic design (Figure 14) 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 SSDs than heuristics, and demonstrate its benefits under two different models.

5.1 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 large sequential and 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 (Figures 4 and 9). In many cases, such events last for 100ms and account for a significant proportion of the device's time, e.g., 50%. Therefore, in mixed workloads, reads may also be blocked considerably, which is prohibitive for many applications. Such blocking is often due to the garbage collector not being able to keep up leading to what is known as internal fragmentation [4].

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. To that end, we look for a general solution that provides predictable performance and minimal latency for reads under mixed workloads. 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 as in [12], may improve the performance under certain workloads and SSD models. However, it does not eliminate the frequent blocking occurring due to garbage collection under a generic workload.

Our design 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_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. We call those two consecutive periods an *epoch*. If D_2 completes syncing and the window is not yet over ($t < T_{min}$), D_2 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 writes of the previous epoch 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.

5.2 Properties and Challenges

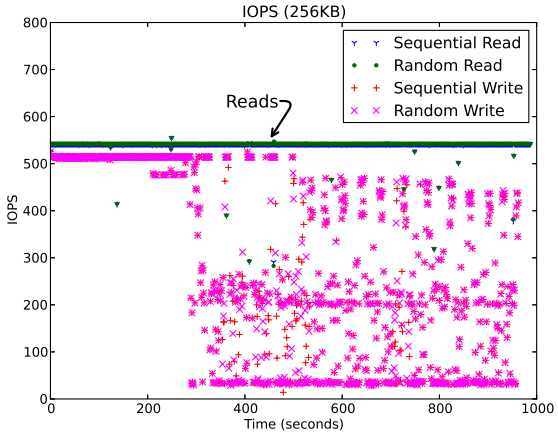
Data consistency & fault-tolerance: By the above design, reads always have access to the latest data, possibly through the cache. This is because the union of the cache with any of the two drives always contains exactly the same (and latest) data. By the same argument, if any of the two drives fail at any point in time, there is no data loss and we continue having access to the latest data. While the system operates with one drive, the performance will be degraded until the failed drive is replaced and the data synced.

Cache size: Assuming we switch drive modes every T seconds and the write throughput of each drive is k MB/s, the cache size has to be at least $2kT$. This is because a total of kT new writes are accepted while performing the previous kT writes to each drive. We may lower that value to an average of $3/2kT$ 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,

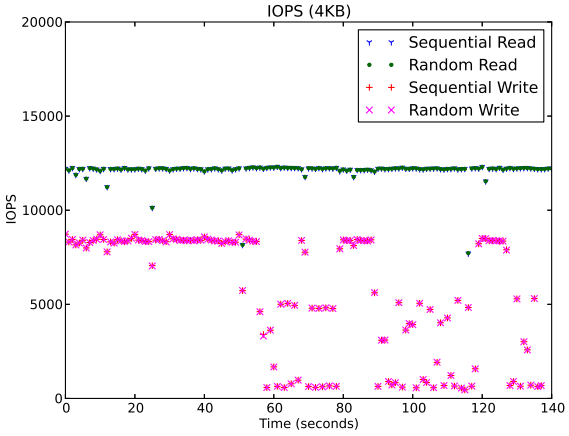
then we need a cache of $2kT = 4000$MBs. The above requires that the drives have the same throughput on average (over T seconds), which is reasonable to assume if T is not small. Moreover, we assume that the rate at which writes are accepted is $k/2$, i.e., half the write throughput of a drive, which is achieved through throttling. That 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 kT 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.

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 kT MBs, which is less than 2GB in the above example. Also, shorter switching periods result in smaller data losses. However, we can avoid data loss by turning incoming writes into synchronous ones. Then the incoming $kT/2$ 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). Moreover, the cache in that case reduces to k. 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. Depending on the system design, other applications may not tolerate a potential data loss. To solve this problem while maintaining close to perfect write stability from the user's perspective and without assuming a battery-backed memory, we propose that writes are committed 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. Instead, we believe it would be best to construct a new 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.

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 and low latency lead to over-provisioning due to the unstable performance of mixed workloads. On the other hand, this design guarantees



(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 15: Physically separating reads from writes leads to a stable and high read performance.

stable performance, which is expected to reduce the amount of over-provisioning. Third, in practice, distributed storage systems already replicate data so the hardware is already available for fault-tolerance and only the software has to be adjusted to apply this design. This requires to generalize our method to distributed systems, which is our next step.

We expect the above design to be applicable on other parts of a system by isolating blocking events (e.g., garbage collection) from cheap operations, e.g., sequential and random requests on hard-drives. Our design may also be used to construct arrays by switching between one set of (possibly) striped read drives and one of writes, or by striping across read/write drive pairs. Finally, we expect this design to also qualify for a new high-performance solid-state drive.

5.3 Preliminary Evaluation

We built a prototype of the above design and verified that it provides high performance, low latency and improved predictability for reads under mixed workloads. For simplicity,

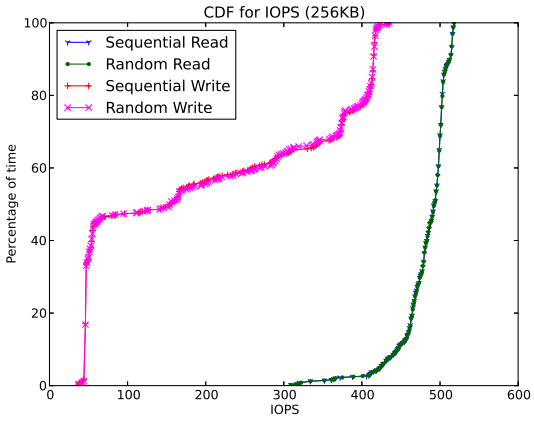


Figure 16: On drive A the read throughput is mostly stable (sequences on the right), with a small variance due to writes after each drive switch.

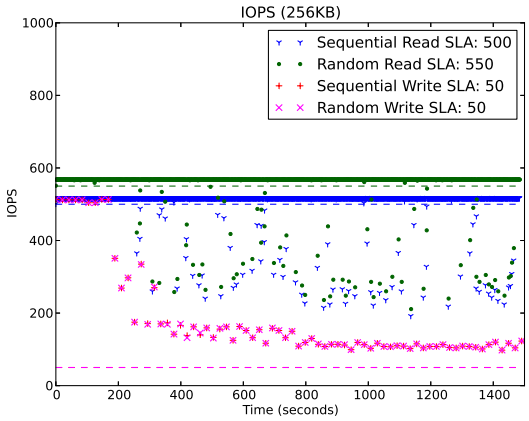


Figure 17: Using our design and QoS method we can provide high guarantees for reads and achieve the targets more than 95% of the time without any heuristics.

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 15(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, on a single SSD and given 50% of the device time, reads are unstable due to writes (Figure 8) and happen at a rate of 525 reads/sec., which is less than half.

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 16 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, as we will later see, when the drive starts reading, the interference is minimal. When providing QoS, that idle time is charged to the write streams, since they are responsible for the blocking. Note that 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.

5.4 Guaranteeing High Performance

Given that we now have stable read performance, it becomes easier to provide high performance guarantees by combining the above method with our QoS method from the previous section. As before, we set four streams each with its own type and throughput requirements. In the following experiments, although the targets of the last two streams are 50 writes/sec, the average throughput plotted is 100 writes/sec or more. That is due to the same writes eventually being dispatched to both drives. Moreover, since the write performance has too much variance at a high granularity, we plot the average write throughput over each epoch (two switch periods) to show that the write guarantees at the drive level are also achieved at the granularity allowed by the device. From Figure 17 we see that under drive B the targets are satisfied 95% of the time. In addition, as shown in Figure 18, by idling writes for a second before each switch the target satisfaction rate increases to more than 99%. In a similar fashion, using drive A we found the same success rates for both the idle and the non-idling case (figures skipped). However, due to the nature of drive A there is a small increase in the variation of the read throughput after each mode switch, which leads to slightly lower guarantees. Since drive B is a newer model one would expect that recent models of similar quality have a behavior closer to that of B. Independently of that, the read performance achieved with any the two drives is significantly more stable compared to that without redundancy. Finally, note that switching less often reduces the throughput drops but requires a larger cache and may leave more background work for the SSD after each switch.

5.5 Evaluation with Traces

In this section we include a short evaluation of our design using the Stg dataset from the MSR Cambridge Traces [11], OLTP traces from a financial institution and traces from a popular search engine [17]. 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 in order to fill the drive cache. Evaluation results using traces can be more challenging to interpret since the traces themselves can be irregular and may have to be studied separately. In terms of read/write isolation, Figure 19(a) shows that reads are affected by writes when using a single drive, while under the same workload our method provides read/write isolation (Figure 19(b)). The write plots for both cases are skipped as they appear as unstable as 19(a).

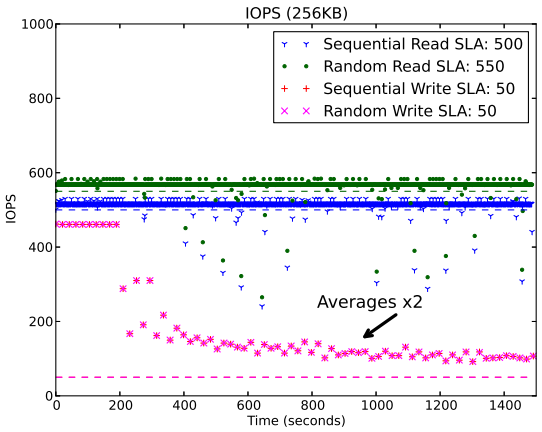


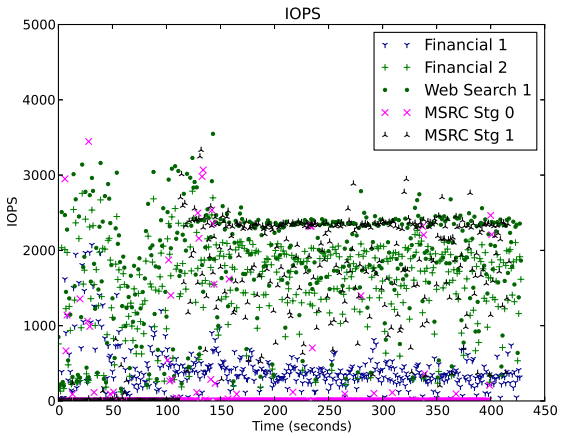
Figure 18: Adding a second of idle time to writes before each switch reduces the read/write interference while satisfying the targets 99% of the time.

6. RELATED WORK

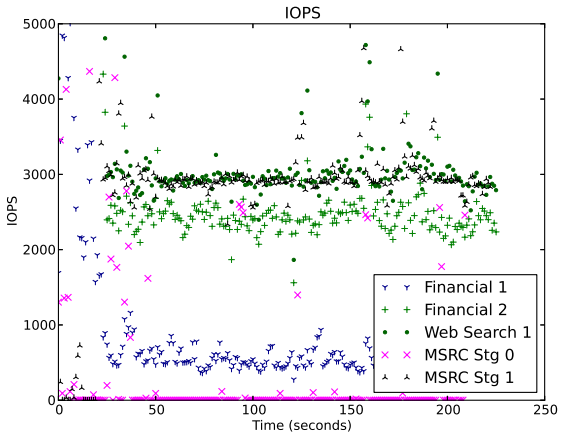
Although the are a number of papers on the performance and characteristics of SSDs, there is little work taking advantage of such results in the context of scheduling and QoS. An example of a fair scheduler optimized for flash is FIOS [12]. According to the results in [12], FIOS is an improvement over OS schedulers that are designed with hard-drive characteristics in mind. Despite that, FIOS has different goals than our work. First, we manage the performance of each client as required rather than providing fair usage. Second, we provide guarantees to the clients rather than best-effort performance. Third, we are interested in constantly minimal read latencies that are significantly less than 100ms. FIOS can give priority to reads in order to decrease the blocking due to writes on certain devices, however, read latencies can still reach 100ms, frequently. Overall, FIOS is not designed to guarantee low latencies but rather as an efficient flash scheduler. In another direction, SFS [10] presents a filesystem designed to improve write performance by turning random writes to sequential ones.

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, [15] 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.

There is a significant amount of work on scheduling and performance management for hard-drive based storage systems, including [8, 9, 13, 14, 16, 18]. Most related to our work are time-based approaches such as Fahrrad [13], which is placed on top of a hard-drive as well as QBox [16], which manages



(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 19: Under a mixture of real workloads our method stabilizes the read performance.

the performance of HDD-based storage systems. Although, the core concepts are similar with respect to scheduling, hard-drives have other characteristics, such as sequentiality and a similar cost for reads and writes. Hence, applying such solutions directly on SSDs leads to low performance, as illustrated in [12] for fair schedulers and discussed in [15].

7. CONCLUSIONS

The performance of solid-state drives degrades significantly under write-heavy workloads. In particular, the average throughput can drop many times while the latency often increases to 100ms due to garbage collection. In this paper, we showed how to provide tight performance guarantees for SSDs at the level and granularity allowed by the drive itself. In addition, we introduced a design based on redundancy that physically separates reads from writes and showed it enables stable high performance for reads under mixed workloads. Finally, by combining the above solutions, we provide high performance and minimal read latency consistently.

8. REFERENCES

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

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

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

[4] 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.

[5] Chen, F., Lee, R., and Zhang, X. 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* (Washington, DC, USA, 2011), HPCA '11, IEEE Computer Society, pp. 266–277.

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

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

[8] Gulati, A., Merchant, A., and Varman, P. J. pclock: an arrival curve based approach for qos guarantees in shared storage systems. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2007), SIGMETRICS '07, ACM, pp. 13–24.

[9] Merchant, A., Uysal, M., Padala, P., Zhu, X., Singhal, S., and Shin, K. Maestro: quality-of-service in large disk arrays. In *Proceedings of the 8th ACM international conference on Autonomic computing* (New York, NY, USA, 2011), ICAC '11, ACM, pp. 245–254.

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

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

[12] 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.

[13] Povzner, A., Kaldewey, T., Brandt, S., Golding, R., Wong, T. M., and Maltzahn, C. Efficient guaranteed disk request scheduling with fahrrad. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008* (New York, NY, USA, 2008), Eurosys '08, ACM, pp. 13–25.

[14] Povzner, A., Sawyer, D., and Brandt, S. Horizon: efficient deadline-driven disk i/o management for distributed storage systems. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing* (New York, NY, USA, 2010), HPDC '10, ACM, pp. 1–12.

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

[16] Skourtis, D., Kato, S., and Brandt, S. Qbox: guaranteeing i/o performance on black box storage systems. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2012), HPDC '12, ACM, pp. 73–84.

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

[18] Wachs, M., Abd-El-Malek, M., Thereska, E., and Ganger, G. R. Argon: performance insulation for shared storage servers. In *Proceedings of the 5th USENIX conference on File and Storage Technologies* (Berkeley, CA, USA, 2007), FAST '07, USENIX Association, pp. 5–5.